Unified Parallel Software Developer's Guide[1]

UPS_VERSION=v-02-07-05

Date of this manual's printing: September 7, 2007

This developer's guide for UPS (Unified Parallel Software) provides the information necessary for developers who are implementing components and internal modules for inclusion in the UPS library. These guidelines ensure the various components of UPS interact properly and in a predictable manner, and ensure the continuity of the software conventions.

The hero model doesn't scale.
M. Peterson

[1] LA-CC 03-041

Contents

1 COPYRIGHT **5**

2 Introduction **6**

3 Getting Started **7**

4 Directory structure **40**
4.1 Main UPS Directory . . . 42
4.2 aux . . . 42
4.3 doc . . . 42
4.4 include . . . 42
4.5 lib . . . 43
4.6 peer_review . . . 43
4.7 script . . . 43
4.8 src . . . 43
4.8.1 Automatic Generation of Interfaces . . . 44
4.8.2 src User API . . . 45
4.8.3 src internals . . . 46
4.8.4 src utils . . . 46
4.9 testing . . . 47
4.10 tools . . . 47

5 Programming Practices **48**
5.1 Variable Name conventions . . . 48
5.2 Style . . . 48
5.2.1 Spacing . . . 49
5.2.2 Comments . . . 49
5.2.3 Common Functionality . . . 50
5.3 Memory Allocation . . . 51
5.4 Error Checking . . . 51
5.5 Environment Variables . . . 51

6 Porting **52**
6.1 Necessary Modifications . . . 52
6.2 Porting to SGI . . . 53
6.2.1 Running on Nirvana and Blue Mountain . . . 54
6.2.2 Running on SGI workstations . . . 55
6.2.3 Problems on SGI . . . 55
6.3 Porting to TFLOP . . . 55
6.3.1 Running on TFLOP . . . 55
6.3.2 Problems on TFLOP . . . 56
6.4 Porting to Linux:naxos . . . 57
6.4.1 Running on Linux:naxos . . . 57
6.4.2 Problems on Linux:naxos . . . 58
6.5 Porting to Linux:lambda . . . 59
6.5.1 Running on Linux:lambda . . . 59
6.5.2 Problems on Linux:lambda . . . 60
6.6 Porting to Linux:intel1/bengal . . . 61
6.6.1 Running on Linux:intel1/bengal . . . 61
6.6.2 Problems on Linux:intel1/bengal . . . 62

6.7 Porting to Compaq Q - Alpha/HP/OSF . . . 62
6.7.1 Running on Compaq Q . . . 62
6.7.2 Problems on Compaq Q . . . 63
6.8 Porting to Sun . . . 64
6.8.1 Running on Sun . . . 64
6.8.2 Problems on Sun . . . 65
6.9 Porting to AIX . . . 65
6.9.1 Running on AIX . . . 66
6.9.2 Problems on AIX . . . 67
6.10 Porting to 68
6.10.1 Running on 68
6.10.2 Problems on 68

7 The UPS CVS Repository **69**
7.1 Version Number . . . 69
7.2 Committing code to the repository and installing/releasing . . . 69

8 Adding Parts **73**
8.1 Adding a New Test . . . 73
8.2 Adding a User Accessible Function . . . 73
8.3 Adding a New Package . . . 74
8.4 Adding New Aux Product . . . 74
8.5 Adding New Tool . . . 75
8.6 Adding Ability for Someone to Access UPS . . . 75

9 How do I **77**
9.1 CVS . . . 77
9.1.1 ...get rid of empty directories and get files that others have a dded . . . 77
9.1.2 ...look at an older version of an existing file with cvs? . . . 77
9.1.3 ...look at a specific version of a file . . . 77
9.1.4 ...look at an old version of a file in a directory that doesn't exist anymore . . . 77
9.1.5 ...check out a previous installation . . . 78
9.1.6 ...add a directory tree to the repository . . . 78
9.1.7 ...remove a directory tree from the repository . . . 78
9.1.8 ...turn off commits . . . 79
9.1.9 ...look at the differences you have made in a file as a whole . . . 79
9.1.10 ...check the differences between a file and the most recent checked in version . . . 79
9.1.11 ...move the whole repository to a new location . . . 79
9.1.12 ...abort a commit . . . 80
9.2 Perl . . . 81
9.2.1 ...edit a whole bunch of files with a simple search/replace . . . 81
9.3 Make . . . 82
9.3.1 ...eliminate most errors I am having with make . . . 82
9.3.2 ...switch from creating a debug version to an optimized version of libups.a . . . 82
9.3.3 ...easily run the tests given that I have 2 libraries . . . 82
9.3.4 ...install ups, but just copy the files . . . 82
9.4 SQA . . . 83
9.4.1 ...get others to review my code I wish to commit . . . 83
9.4.2 ...use Insure (compile-time/run-time code checker) . . . 83
9.4.3 ...use Purify (run-time code checker) . . . 84
9.5 Xemacs . . . 85

9.5.1 ...change file type mode . . . 85
9.6 Misc . . . 86
9.6.1 ...recover files from a NetApps file system . . . 86
9.6.2 ...modify the ups-team/ups-user mailing lists . . . 86

A Source Examples **88**
A.1 Makefile Example . . . 88
A.2 C Include File Example . . . 90
A.3 C Source Example . . . 92

List of Figures

1 Example of CVS Commit Log . . . 71

1 COPYRIGHT

2 Introduction

UPS, an acronym for "Unified Parallel Software", is a library of routines designed to help the application developer create efficient, extensible, and robust large scale parallel programs for physics simulations. It is designed to run in any computing environment that supports the C programming language and which provides a method for moving data between parallel processes[2].

Some parallel programming models attempt to hide the parallelism from the application writer, while others require the application writer work at the lowest levels. UPS falls in between: it is designed to expose the parallel environment in a natural way while abstracting away the complexities.

This developer's guide provides programmers the details necessary for contributing to the UPS software library. Such contributions may be in the form of entire components or as software internal to a component. For an overview of UPS from the application developer's perspective, see the User Guide [1].

We expect three steps to be followed by potential UPS developers:

1. read the user guide
2. communicate with UPS team, and
3. read this developers guide.

[2]The default mechanism is MPI[3].

3 Getting Started

The following is the README file found in the top level directory.

```
$Id: README,v 1.109 2005/10/14 20:17:54 lmdm Exp $

LA-CC 03-041, C-03,097

This file contains general information on how to build UPS and the
change history.

If you get stuck, contact ups-team@lanl.gov.  Feedback is appreciated as well.

Also, for more information see
    doc/UserGuide/UserGuide.ps and
    doc/DeveloperGuide/DeveloperGuide.ps

    http://public.lanl.gov/ups

----------
 Sections
----------
- Basic Build Overview
- Necessary Utilities
- Creating make.inc
- Personal Workspace Build: out-of-tree build
- Personal Workspace Build: build then mini-install
- Installation Build
- Warnings/Errors
- Expected times
- Tags
- Outstanding Issues
- Release Messages
----------------------
 Basic Build Overview
----------------------
This section gives a condensed version of what you should probably do if
given a UPS tar file.  If you wish to do something special, read the
sections below for more information.

  1) % gzip -dc UPS_dist_<YYMMDD>.tar.gz | tar xvfp -
     This will gunzip/untar the UPS distribution tar file you were given.
  2) % cd UPS
     This is the directory that is created by the above command.
  3) % ./configure.pl go -debug INSTALL_DIR=/tmp/ups_instal
     This configures without debug (eg optimized) and sets the installation
     directory to be /tmp/ups_install (so change that to be whatever).
  4) % gmake
     This builds UPS and runs the tests.
  5) % gmake install_arch
     This installs into the directory specified by INSTALL_DIR above.
```

```
You can type in "gmake help" for a list/explanation of targets.

---------------------
 Necessary Utilities
---------------------
Hopefully, everything is set up and you won't need to change anything.
However, one should check this section to assure everything is accessible.
There are different levels of "building UPS".  If you are not interested,
for example, in building the Guides, you do not need to have access to
the utilities listed in number 4 (below).

It makes life easier if the utilities are in your path.  UPS is designed to
take advantage of the PATH environment variable.  In some cases (eg the CC
variable in configure.dat/make.inc), you can specify a the full path name
to the utility.

1) Requirements for building UPS - this is what is minimally needed:
   0- gmake
   0- perl
   0- C compiler
      Set in configure.dat/make.inc (CC and FLAGS_*_C variables)
      If you are building without Fortran, you must give the
      "+no_fortran" flag to configure.pl.
      See Creating make.inc below.
   0- MPI
      UPS must be able to find the location of mpi files.  On some systems
      (eg bluemountain), MPI is tightly integrated with the system and
      the compiler will find the include file without having to specify
      an additional "-I<location>" flag.  On other systems (especially
      if you are using mpich), set the MPI_ROOT environment variable
      to point to the correct location (eg ${MPI_ROOT}/include points to
      the directory containing mpi.h).
      See Creating make.inc below.

2) Requirements for running UPS tests:
   0- MPI
      UPS must be able to run executables.  Modify RUN_NAME in
      configure.dat/make.inc if necessary.

3) Requirements for building UPS with Fortran:
   0- Fortran compiler
      Set in configure.dat/make.inc (FC and FLAGS_*_F variables).
      See Creating make.inc below.

4) Requirements for building Guides:
   0- latex
      Our documents are written in LaTeX format.
      latex:    builds .dvi file from .tex files
      dvips:    builds .ps file from .dvi file
      pdflatex: builds .pdf file from .tex files
      These executables are located on (if mounted):
```

```
           /usr/projects/ups/teTeX/bin/mips-irix6

5) Requirements for Script/ups_aa_statistics_plot.pl:
   0- gnuplot
      This script uses gnuplot to create a post-script document from
      ups_log.txt.  Again, if you are not interested in running this
      script, you do not need access to gnuplot.

6) Requirements for CVS access to repository
   0- cvs - version 1.10 or later
   0- Scripts: some useful scripts are located in the CVSROOT product.
        So, you need to set your path to $CVSROOT/CVSROOT or check out
        CVSROOT and set your path to that directory.
   0- In order to get UPS from the repository, one can:
           cvs -d /usr/projects/ups/Repository co -P UPS

--------------------
 Creating make.inc
--------------------
Run the configure script:

   configure.pl go

configure.pl uses configure.dat to create make.inc.

make.inc is a gmake include file containing variables, suffix rules,
targets/prerequisites/rules, and other gmake commands.  The variables
are defined according to what the value of CONFIGURE_OPTIONS is
in configure.dat.  If you wish to make permanent changes to either the
CONFIGURE_OPTIONS or the gmake variables, you can edit configure.dat.
However, the options and variables can be overridden from the command line
of configure.pl so this might be an easier solution.  Of course, you
may modify make.inc itself.  However, changes done will be overridden
when configure.pl is run again.

The directory where configure.pl creates make.inc is the following (in order
of preference):
        o "-d <directory>" flag passed to configure.pl
        o current working directory is the default

Type in "configure.pl" to get info about the basics of the configure.pl script.
Type in "configure.pl help" for common use cases.

We have tried to make it so no modification of files is needed.  One should
only have to type in "configure.pl go" in order to get things to run.

I have the following code in my .cshrc to help automate this process:

    switch ("`uname -n`")
    #...if you are on lambda...
    case lambda:
```

```
    case l[0-9][0-9]:
    case l[0-9]:
        setenv UPS_PREFIX /netscratch/${USER}/UPS_build
        setenv PATH ${PATH}:${UPS_PREFIX}/bin/LINUX32_mpi
        setenv LD_LIBRARY_PATH \
            ${LD_LIBRARY_PATH}:${UPS_PREFIX}/lib/LINUX32_mpi
        breaksw
    #...if you are on theta...
    case theta:
    case t[0-9][0-9]:
    case t[0-9]:
        setenv UPS_PREFIX /scratch/${USER}/UPS_build
        setenv PATH ${PATH}:${UPS_PREFIX}/bin/SGI64_mpi
        setenv LD_LIBRARY64_PATH \
            ${LD_LIBRARY64_PATH}:${UPS_PREFIX}/lib/LINUX32_mpi
        breaksw
    #...default...
    default:
        setenv UPS_PREFIX /scratch/${USER}/UPS_build
        breaksw
    endsw
    alias configups 'configure.pl -d ${UPS_PREFIX}'

Now, after running the aliased "configups go [options]" command, I don't
have to worry about it any more.  The make.inc file is placed in the
${UPS_PREFIX} directory along with the build files (see "Personal Workspace
Build" below).

-------------------------------------------
 Personal Workspace Build: out-of-tree build
-------------------------------------------
 Suppose you have the UPS source on a cross mounted directory
 and are building it for different architectures.  Also, suppose that the
 arguments passed to configure.pl are the same for both architectures.
 An out-of-tree build will simplify builds.

 - UPS source: /home/lmdm/UPS
 - Machines:   A and B
 - Environment Variables: UPS_PREFIX = top dir for architecture dependent files
                          UDM_ROOT = where UDM is located
 - Machine A configure.pl options: -debug +no_fortran
 - Machine B configure.pl options: +i8r8

 1) machine A .cshrc: setenv UPS_PREFIX /netscratch/lmdm/UPS_build
                      setenv UDM_ROOT /foo/bar/UDM_linux
    machine B .cshrc: setenv UPS_PREFIX /scratch/lmdm/UPS_build
                      setenv UDM_ROOT /foo/bar/UDM_irix64
    Basically, set these variables in your .cshrc so they are the
    correct values for what machine you are currently on.
 2) machine A: configure.pl -d ${UPS_PREFIX} go -debug +no_fortran
    Initial creation of ${UPS_PREFIX}/make.inc.
```

```
 3) machine A: gmake
    Build and put architecture dependent stuff in /netscratch/lmdm/UPS_build
    on machine A using the UDM at /foo/bar/UDM_linux
    This will build the documentation (if necessary), build the ups
    library(ies), build the tests, and run the tests.
 4) machine B: configure.pl -d ${UPS_PREFIX} go +i8r8
    Initial creation of ${UPS_PREFIX}/make.inc.
    As in step 2, this creates ${UPS_PREFIX}/make.inc
 5) machine B: gmake
    Build and put architecture dependent stuff in /scratch/lmdm/UPS_build
    on machine B using the UDM at /foo/bar/UDM_irix64
    This will build the documentation (if necessary), build the ups
    library(ies), build the tests, and run the tests.
 6) machine ?: gmake
    From then on, when you go to a different machine, you no longer have
    to change make.inc - just run your gmake command.

 Concurrent builds (gmake on machine A while a gmake on machine B is
 running) are possible since all machine dependent files are located
 in their own directory (${UPS_PREFIX}).

-----------------------------------------------------
 Personal Workspace Build: build then mini-install
-----------------------------------------------------
1) create make.inc (see Creating make.inc above)
   One can override the default installation directory by configuring via:
        configure.pl go INSTALL_DIR=/tmp/my_temp_install
   Other variables that may be set as well are:
        INSTALL_DIR_BIN
        INSTALL_DIR_DOC
        INSTALL_DIR_INCLUDE
        INSTALL_DIR_LIB
        INSTALL_DIR_SCRIPT
2) gmake
3) gmake install_arch
   This last step just copies files to the INSTALL_DIR location.
   This does not need to be done if you are just going to be
   using the files in their original locations.

-------------------------------------------
 Official Installation Build for Developers
-------------------------------------------
1) create make.inc (see Creating make.inc above)
2) gmake full
   This does a gmake for different configure.pl command line settings
   and builds necessary documentation if necessary.  If you were given
   a distribute tar file, all the documentation is already built.
3) gmake install_dev
   This copies over files to the INSTALL_DIR/PRODUCT_VERSION, installs
   the web pages, and does a cvs tag (if cvs accessible).
   If you were given a distribute tar file, you probably want to do a
```

```
   "gmake install_arch INSTALL_DIR=<location>" instead.

-----------------
 Warnings/Errors
-----------------
After building UPS (ie running gmake), you can type in
   gmake check_all
This will filter the output file ($UPS_PREFIX/make_all_<arch>.txt) and
return any errors/warnings/remarks that are important).

The following are warnings/errors you might/should see (which are O.K.):

  --BUILDING--
1)      **Error** Needed to update [make.inc].
          ...
          Now re-run your gmake command.
   make.inc is automatically generated and sometimes needs to be
   rebuilt (eg if you modify configure.dat).  The build system detects
   this and remakes make.inc.  You then need to re-run your gmake command.
2)      <LaTeX warnings>
   If the Guides need to be rebuilt, LaTeX will be run and it spits out
   warning messages like there was no tomorrow (thousands of lines).
   This is normal for latex and eventually, the Guides should be built.
3)      ar: Warning: creating ...
   ar gives a warning when a library is created.
4)      ld64: WARNING 84 : ...lib... is not used for resolving
   Some additional libraries are included that might not be needed for
   certain executables.

  --TESTING--
1)      **UPS Error** upsp_er_sig_alarm (UPS detected hang) (-997)
   Testing of alarms in the er package are supposed to generate this message.

After all the tests have run, a pass/fail summary is printed.  All tests
should pass.

  --INSTALLING--
1)      tar: can't set time on .: Not owner
   When installing the web pages, one might see this error message.
   This is ok.

----------------
 Expected times
----------------
I have given the "linux" times.  If you are on a slower machine (as there
is no faster machine than linux :) ), you might have a time multiplier.
For example, on a busy sgi (theta), multiply the time by 5.

- HDF (an aux product UPS uses): 5 minutes
- UPS lib/execs:                 5 minutes
- UPS Running Tests:             2 minutes
```

```
------
 Tags
------
  v-##-##-##:   full release
  v-##-##-##_*: minor release before v-##-##-## release
  sf:           original tar bal put into SourceForge UPS project
  glp_deposit:  what was given to business folks for gnu open licensing

------------------
Outstanding Issues
------------------
  - LAMPI: When running UPS tests with lampi, the dp sort test
    fails due to LAMPI bug.  MPI_Alltoallv should allow for NULL
    buffers if the counts are 0 but it doesn't.
    Reported to LAMPI folks and will be fixed.
    No new UPS needs to be built - just relink of executable.

================================================================================
                            RELEASE MESSAGES BELOW
================================================================================

==========================
New UPS Version v-02-07-04 (2005/10/14)
==========================

OS Tested:
   IRIX64
   Linux (and with bproc)
   OSF1
   AIX

Environment:
   Please see lib/libups.settings for the environment used.

Products:
        hdf: hdf5-1.6.2

Please email ups-team@lanl.gov if you have any questions/comments.

Major Changes since v-02-07-03 (apart from various bug fixes)
==============================
o DP Sort
  Changed sorting technique from a bubble sort to a binning sort:
    . Sample to create bins
    . Disperse data to pes according to bins
    . PEs sort on local data
    . PEs disperse data back to original sizes
  This sort takes up more memory (about 2-3x original buffer size)...but
  is much faster.  On pink (bproc linux cluster), it sorted 2 billion doubles
  distributed to 128 PEs in 1 minute (about 10000 times faster than the
```

```
  original method).

o CM Shared Memory
  - Added ability to set the process group that can connect to a shared
    memory area via UPS_CM_SM_set_item or the UPS_AA_MEM_ITEM_P_GROUP.
    By default, the value is UPS_CM_P_GROUP_BOX (same hostname).
    You might, for example, wish to see how your algorithm performs using
    no shared memory and set it to UPS_CM_P_GROUP_SELF.

==========================
New UPS Version v-02-07-03 (2004/06/12 00:29:09)
==========================

Machines Installed: theta/bluemountain, lambda, q(s)

Environment:
osf:   fortran_5.5.1, CXX_6.5.1, MPI_64bit_Thread_Safe_R13.4
sgi:   MIPSpro_7.3.1.2m, mpt_1.6
linux: absoft_8.2, gcc_3.2.3,
               mpich_1.2.5.absoft_8.2

Products:
        hdf: hdf5-1.6.1

Please email ups-team@lanl.gov if you have any questions/comments.

Major Changes since v-02-07-02 (apart from various bug fixes)
============================
o Ported to lightning cluster

o GS and IO Package
  Added better statistics (eg MB/S rates) that are printed out in
  ups_log.txt output file.

o check_space target added
  When building UPS from source, checks are periodically made to see
  if enough space is available.  A warning gets printed if there might
  not be sufficient space.
  Type 'gmake help' for more info.

o HDF version change from hdf5-1.4.5-post2 to hdf5-1.6.1
  This is transparent to the users.

o Gather/Scatter Package

  + If given a non-0 gs_id value during a setup call, UPS will try and use
    that ID value (will assign another value if necessary).

  + If given a 0 gs_id value during UPS_GS_Free, UPS will free all gs
    structures built from gs setup calls.
```

```
+ Added ability to get/set GS optimization options more easily (and see
  how effective the optimizations were).  Nothing new is needed to be
  done by the user - these just make it easier to changes options if
  desired.

   - The following options that affect GS optimizations have been added
     to the UPS_AA_Opt_get() call (any may be set as environment variables as
     well):
     UPS_GS_OPT_COMPRESSION, /* Purpose: sets whether or not to try compression
                              * Datatype: int4 (UPS_DT_TRUE=1/UPS_DT_FALSE=0)
                              * Get: yes
                              * Set: yes
                              * If true, will try to use compression.
                              * (a duplicated index is sent once and coppied
                              * on the receiving side) */
     UPS_GS_OPT_COMP_LOOKAHEAD, /* Purpose: num of indices for compression scan
                                 * Datatype: int8
                                 * Get: yes
                                 * Set: yes
                                 * This sets the number of indices to look for
                                 * when doing compression.  The different
                                 * values are:
                                 *   <0: scan whole array (costly)
                                 *    0: do not do compression
                                 *   >0: look ahead this number of indices */
     UPS_GS_OPT_COMP_RATIO, /* Purpose: sets minimum compression ratio
                             * Datatype: real8
                             * Get: yes
                             * Set: yes
                             * Ratio = 100*Comp_Size/UnComp_Size (rounded)
                             * Values will then ranbe from 0 - 100.
                             * The lower the value, the more compressed.
                             * The ratio for a particular GS setup must be less
                             * than this value in order to use compression.
                             *   <0: Use compression for any ratio
                             *    0: do not do compression
                             *   >0: use this ratio value */
     UPS_GS_OPT_COMM_ORDERING, /* Purpose: num of indices for compression scan
                                * Datatype: int4 (UPS_DT_TRUE=1/UPS_DT_FALSE=0)
                                * Get: yes
                                * Set: yes
                                * If true, will try to order send/recvs
                                * (eg start offbox communication before onbox)*/

     NOTE: all processes must set the same values

   - The following option has been added to the UPS_GS_Get_item() call
     to get the amount of compression that was done for a particular
     setup GS id:
     UPS_GS_COMP_EFFECT: real8 ranging from 0-1 with 0 meaning no
       compression.
```

```
==========================
New UPS Version v-02-07-02
==========================

Machines Installed: theta/bluemountain, lambda, q(s)

Environment:
osf:   fortran_5.5.0, CXX_6.5.1, MPI_64bit_Thread_Safe_R12
sgi:   MIPSpro_7.3.1.2m, mpt_1.6
linux: LaheyFortran95Pro_6.1, gcc_3.2.1,
               mpich_1.2.4.LaheyFortran95Pro-6.1

Products:
        hdf: hdf5-1.4.5-post2

Please email ups-team@lanl.gov if you have any questions/comments.

Major Changes since v-02-07-01 (apart from various bug fixes)
=============================
o UPS Internal Error Reporting
  One can call UPS_AA_Opt_set( UPS_ER_OPT_OUTPUT, &output_type ) and
  set output_type to one of the following
        UPS_ER_OUTPUT_DEFAULT (standart error)
        UPS_ER_OUTPUT_FILE    (ups_err.txt file)
        UPS_ER_OUTPUT_NONE    (no output)

  See the Use Details:Error Reporting section of the UserGuide for
  more information.

o UPS Version Consistency

  More ability to see/check UPS version for consistency has been added.
  Now users can get the UPS version number from the following:
  - Installation Directory (v-##-##-##)
  - Header Files (ups.h, upsf.h, upsf77.h)
    UPS_VERSION constant defined
  - Function Call
    F:   call UPSF_AA_OPT_SET( UPS_AA_OPT_VERSION_CHECK, UPS_VERSION, ierr )
    F77: call UPS_AA_OPT_SET(  UPS_AA_OPT_VERSION_CHECK, UPS_VERSION, ierr )
    C:   int version_ups=UPS_VERSION;
         UPS_AA_Opt_set( UPS_AA_OPT_VERSION_CHECK, &version_ups );
  - Function in libups.a (ups_version_<version number>)
    Users can 'nm libups.a | grep -i ups_version' to get the version of
    the library.
    C users may make a call to UPS_VERSION_CHECK() which will show a
    version mismatch at link time (as opposed to run time).
    C:   UPS_VERSION_CHECK();
         This is an additional call to C users that will show

  See the Use Details:UPS Version Consistency section of the UserGuide for
```

```
  more information.

o GS Package - Added two items that can be gotten from UPS_GS_Get_item()
  - UPS_GS_INDEX_PE
    (int array) The size of this array can be obtained by
    UPS_GS_NUM_GLOBAL_INDICES.
    Regardless of which GS setup call made, UPS_GS_INDEX_PE and
    UPS_GS_INDEX_VALUE can be obtained.  The values of these
    correspond to the index_pe and index_value arguments to
    UPS_GS_Setup_s_local() call.  See that function for more
    information.
    Each value of this array contains the process number that owns
    the corresponding index.
    Possible values:
       UPS_GS_INDEX_ZERO:  scatter skips, gather zeros
       UPS_GS_INDEX_SKIP:  scatter skips, gather skips
       0 through numpes-1: process the index resides on
  - UPS_GS_INDEX_VALUE
    (int array) The size of this array can be obtained by
    UPS_GS_NUM_GLOBAL_INDICES.
    Regardless of which GS setup call made, UPS_GS_INDEX_PE and
    UPS_GS_INDEX_VALUE can be obtained.  The values of these
    correspond to the index_pe and index_value arguments to
    UPS_GS_Setup_s_local() call.  See that function for more
    information.
    Each value of this array contains the 0 based index into the
    local array of the process that owns the corresponding index.
    If the corresponding index_pe is a special index (less than 0),
    the corresponding index_value is not used.
    Possible values:
       UPS_GS_INDEX_ZERO:  scatter skips, gather zeros
       UPS_GS_INDEX_SKIP:  scatter skips, gather skips
       >=0:                0 based index into local array

o AA Package - Initializing UPS from C or Fortran Main Programs

  [The default behavior is still the same (ie if you have working code now,
  you do not have to change anything).]

  Previously, if the main program was C, you had to use the C UPS
  initialization routine UPS_AA_Init(argc, argv).  Likewise, if the
  main program was Fortran you had to call the UPS Fortran interface
  initialization routines UPS_AA_INIT(error_flag) or
  UPSF_AA_INIT(error_flag).  This is due to a quirk of MPI where the
  appropriate language interface to MPI_Init must be called.

  Some flexability has been added so now users can set options (via
  calling to UPS_AA_Opt_set) telling UPS what the main language is (and
  what the command line arguments are if using C).  This allows, for
  example a single call to UPS_AA_INIT (Fortran UPS initialization call) to
  be used for Fortran main programs (default) and C main programs (as long
```

```
  as the C main calls UPS_AA_Opt_set and sets the language and command line
  options correct options).

  See the UPS_AA_Init in the UserGuide for examples of use.

o tools/libuserd-HDF: ups_libuserd_script.pl
  This script is used to create casefile(s) and an EnSight input file.
  This input file can then be fed to EnSight to load up all meshes
  found in the groups specified by the -d option:
        % ups_libuserd_script.pl gresho.pst.0000* -d /jade
        % ensight7 -p ups_libuserd.script

o For Building UPS: New "module" system.
  A new module system has been installed.  This required a small change
  to the Makefile system (one line in configure.dat).

o For Building UPS: Software Quality Tools
  I have added the ability to more easily run tools like insure, gcov, lint,
  third, mview, ... .  Run 'configure.pl help' for more info.

==========================
New UPS Version v-02-07-01
==========================

Machines Installed: theta/bluemountain, lambda, q(s)

Environment:
osf:   fortran_5.4.1, CXX_6.5.1, MPI_64bit_Thread_Safe_R12
sgi:   MIPSpro_7.3.1.2m, mpt_1.6
linux: LaheyFortran95Pro_6.1, gcc_3.0.4,
               mpich_1.2.4.LaheyFortran95Pro-6.1

Please email ups-team@lanl.gov if you have any questions/comments.

Major Changes since v-02-07-00 (apart from various bug fixes)
=============================
o tools/libuserd-HDF
  - Setting environment variable LIBUSERD_HDF_SET_SOLUTION_TIMES
    Setting this causes the reader to set solution times itself
    instead of using the one found in the hdf file.  The solution
    time will be the same as the time step.  This is useful if
    you wish to look at multiple meshes that have the same
    solution time (by default, data with the same solution time
    is ignored...a requirement of EnSight).
o tools/ups_libuserd_prep.pl
  - added -v flag for verbose output
o HDF
  New version of HDF used - hdf5-1.4.5-post2

==========================
New UPS Version v-02-07-00
```

```
=========================

Machines Installed: theta/bluemountain, lambda, q(s)

Environment:
osf:   fortran_default, CXX_default, MPI_64bit_R12
sgi:   MIPSpro_default, mpt_default
linux: LaheyFortran95Pro_6.1, gcc_3.0.4,
               mpich_1.2.4.LaheyFortran95Pro-6.1

Please email ups-team@lanl.gov if you have any questions/comments.

Major Changes since v-02-06-00 (apart from various bug fixes)
==============================
o Port to LLNL AIX compute servers
  UPS has been ported (well, re-reported) to open and secure LLNL
  compute servers (blue/frost and white).  I will periodically install
  new versions on those machines now as well.

o UT Package: UPS_UT_Sleep()

  A high resolution sleep has been added to UPS.  This portable function
  takes a double for the seconds to sleep.  nanosleep or usleep is called
  on supporting architectures.

o IO package: added optioins to UPS_IO_Info_item_get()

  UPS_IO_INFO_PATH and UPS_IO_INFO_PATH_LENGTH will return info
  about the full path to the object in the info_id obtained
  from UPS_IO_Info_create()/UPS_IO_Info_create_self() calls.
  The full path to an object is nice since you can point to it
  from any location id - something you cannot do with a relative
  path (ie UPS_IO_INFO_NAME).

o Tools
  bin/ups_io_bob_to_hdf
         This tool takes a 'brick of bytes' and writes them to an HDF file.
         This HDF file can the be visualized via the EnSight reader
         libuserd-HDF.
  lib/libuserd-HDF
       - Added ability for user to supply graphics files where the number
         of variables and/or parts can change from time step to time step
         (EnSight by itself cannot handle this).

         However, there is a performance detriment at startup for this
         ability.  If possible, you should consider using the following
         environment variables (from the README):

           - LIBUSERD_HDF_CONSISTENT_PARTS
             LIBUSERD_HDF_CONSISTENT_VARS
             This comma delimited list (no whitespace) contains the time
```

```
            steps in which parts and/or vars change.
            By default (environment variables not being set), the reader
            assumes that the following can change from time step to
            time step:
               PARTS:
                 o Number of parts
                 o Path to each part in HDF file
                 o Structured:   ijk dimensions remain the same
                   Unstructured: number of elements for each element type
               VARS:
                 o Number of variables
                 o Path to each variable in HDF file
            Significant performance at EnSight startup can be obtained if
            you can set these environment variables.
            Example: setenv LIBUSERD_HDF_CONSISTENT_PARTS
                     Parts remain consistent throughout the time steps.
            Example: setenv LIBUSERD_HDF_CONSISTENT_PARTS 0,5
                     Times steps 0 through 4 are consistent and time steps
                     5 on are consistent.
            If unsure which time steps are consistent for parts or vars,
            use 'ups_libuserd_prep.pl <casefile>'.

          Also more statistics were added to the reader.  The compute-intensive
          portions of the reader are statistics-ized and the final output
          file ups_log.txt has this info (first 10 code locations).

        - Added ability for HDF-EnSight reader to look in multiple places
          for EnSight geometry and/or variable information.  This
          is done by creating additional top level string attributes that
          start with the name EnSight_model.  The values of these attributes
          are the full path to locations in the file to look for EnSight data.

          If duplicate data is found, the first one will be used (eg duplicate
          variable names).
  bin/ups_aa_libuserd_query to bin/ups_libuserd_query
        Can now get a lot more information about the file.
        Run this tool without arguments to get a help message.
  bin/ups_libuserd_prep.pl
        Script to help in EnSight reader performance.
        Run this tool without arguments to get a help message.
  ups_procmon
        A simple process monitor has been written that gives information
        about a process (via a time-sampling technique).
        This tool is still under development...and under review to see
        if it is useful.
        See the documentation in doc/doc_ups_procmon for more information.

o Performance Enhancements
  - Tools/libuserd-HDF EnSight reader
  - Default behavior for UPS internal memory checking has been changed
    from:
```

```
       always on
      to
       on if UPS built in debug mode and off otherwise
    You can still modify this behavior by setting UPS_MEM_NO_MANAGEMENT
    environment variable.  Set to 0 if you want memory management on,
    set to nothing or non 0 if you do not want memory management.
  - Removed eliminated additional creat in UPS_IO_Info_count

o Building without some UPS packages
  Although the default is to build/install UPS with all of its pacakges
  built, it is now possible to build UPS without some of its higher level
  packages(DP = Data Parallel, IO = File IO, and/or GS = Gather Scatter).
  If there are any requests to do so, I can build UPS this way.

o Profiling Libraries
  UPS can now be built under varios profiling libraries (insure++, lint,
  vampir, fpmpi, ...).

o HDF
  UPS now uses (and builds) the new release hdf5-1.4.5 .

==========================
New UPS Version v-02-06-00
==========================

Machines Installed: theta, lambda, bluemountain, q(s)

Please email ups-team@lanl.gov if you have any questions/comments.

Major Changes since v-02-05-02
==============================
o Tools
  bin/ups_aa_libuserd_query
        Purpose: Query EnSight files using a USERD defined lib
        (eg libuserd-HDF.a).

        Format:
          ups_aa_libuserd_query -casefile <casefile> [option(s)]

        Options:
          -casefile: Whatever you would supply to EnSight in the
              'Set (Case)' field when defining the data reader.
              You might have to run ups_aa_libuserd_query in the
              same directory as the casefile.
          -v: verbose output (screen output and ups_log.txt file
              created)
         -number_of_time_steps
         -solution_times

        Examples:
          - ups_aa_libuserd_query -casefile foo.h5 -solution_times
```

```
           Return the solution times for the foo.h5 casefile.

  lib/libuserd-HDF
        Added ability for user data files to be 2-d and still be
        understandable to EnSight.  Basically, I simulate dimensions
        until 3-d requirement is met.

  lib/libuserd-HDF
        Added ability to more easily view 'cell' variables for structured
        meshes.  If you use the following methods:

          (coordinates_x, coordinates_y, coordinates_z) or
          (coordinates_axis_x, coordinates_axis_y, coordinates_axis_z)

        An EnSight part for cells will automatically be created:
                part_ups_cell.
        You can then just define variables on part part_ups_cell:
           1d --> bar2
           2d --> quad4
           3d --> hexa8

o UPS General change
         In an effort to reduce the size of UPS
         (documentation/libraries/source code), I am removing functions that
         fall into the following categories:

             - Functionality exists in MPI calls and UPS offers no benefit
             - Functionality exists in other UPS calls

         There are some UPS functions that do fall into the above categories
         that I am keeping.  For example, UPS_CM_Get_penum is commonly used
         and UPS_CM_Bcast allows broadcasting of large messages where most MPI
         implementations would choke on.

         With the removal of unnecessary(?) page-breaks and examples, the
         number of pages in the UPS UserGuide has been reduced by 1/3.

         The following functions have been removed (the substitutions are
         listed).  Email ups-team@lanl.gov if you have any concerns.

      -- send/recv calls
         use MPI_Send, MPI_Recv, MPI_Wait
         to be removed:
         UPS_CM_CINIT
         UPS_CM_CRECV
         UPS_CM_CSEND
         UPS_CM_CWAIT
         UPS_CM_RECV
         UPS_CM_SEND
      -- Packing calls
         use MPI_Pack, MPI_Unpack, MPI_Msginfo, MPI_Probe, MPI_Get_count
```

```
        to be removed:
        UPS_CM_FREE_BUFFER
        UPS_CM_INITSEND
        UPS_CM_MSGINFO
        UPS_CM_PACK
        UPS_CM_PBCAST
        UPS_CM_PRECV
        UPS_CM_PROBE
        UPS_CM_PSEND
        UPS_CM_UNPACK
     -- Collective maxloc/minloc
        use MPI_Allreduce
        to be removed:
        UPS_CM_GLOBAL_MAXLOC
        UPS_CM_GLOBAL_MINLOC
     -- Sendrecv
        use MPI_Sendrecv
        to be removed:
        UPS_CM_SENDRECV
        UPS_CM_SENDRECV_INIT
        UPS_CM_SENDRECV_TWO
        UPS_CM_SR_RESET
     -- Host information
        already contained in UPS_CM_P_group_item
        to be removed:
        UPS_CM_GET_HOST_NUMPES
        UPS_CM_GET_HOST_PENUM
        UPS_CM_GET_HOSTNUM
        UPS_CM_GET_NUMHOSTS
     -- Getting name of datatype
        use UPS_UT_Get_name_or_value
        to be removed:
        UPS_DT_TO_CHAR

o AA Package
        Added the following options that can be used in
        UPS_AA_Opt_get():
        - UPS_AA_OPT_INITIALIZED: returns if UPS has been initialized
        - UPS_CM_OPT_PROTOCOL_INIT: determine if the underlying communication
                protocol (usually MPI) has been initialized.

o IO Package - Creating a Dataset from Pieces

  There may be times when you wish to create a global dataset but you
  only have access to a subset of the pieces at one time.  You can call
  UPS_IO_Dataset_write() multiple times with the following caveats:

  - UPS_IO_Dataset_write() is a collective call.

     All processes must still call UPS_IO_Dataset_write.  If a process
     has no data, set UPS_IO_INFO_DIMS to 0s.
```

```
  - You must set UPS_IO_INFO_NDIMS, UPS_IO_INFO_DIMS,
        UPS_IO_INFO_STARTS and UPS_IO_INFO_DIMS_TOTAL.

     You cannot use the default or UPS_IO_INFO_PGRID settings and you
     must know the total size of the dataset beforehand.

  - The datatype, UPS_IO_INFO_NDIMS, UPS_IO_INFO_DIMS_TOTAL,
        cannot change.

  - You must set UPS_IO_INFO_DIMS and UPS_IO_INFO_STARTS
        when reading the dataset back.

     Saying "must" is a little strong..."should" perhaps is better.
     Using the underlying protocol UPS_IO_PROTOCOL_HDF, the default
     values gotten from UPS_IO_Info_create() ("what this process
     wrote is what it reads") will get each process only the data it
     wrote during the last UPS_IO_Dataset_write().  However, other
     protocols might behave differently.

     So, better safe than sorry - set dims and starts manually.

o IO Package - Recursive info count/create
         Added the ability to for UPS_IO_Info_count and UPS_IO_Info_create
         to work recursively via calling UPS_IO_Filter_set to set the
         appripriate filter.  Previously the count/create calls would
         only get information about the current object (eg the number
         of members in a group).

         Currently, the only filters available are the following:
            - object_name
              This will do a match for object_name in the
              members in the list.
              This effectively limits the number of
              matches to 1 (if found) or 0 (if not found).
            - */object_name
              This will also match object_name
              recursively through any subgroups.
              Not valid for UPS_IO_INFO_LIST_ATTRIBUTES.
            - */
              This will do a recursive listing of all
              members.
              Not valid for UPS_IO_INFO_LIST_ATTRIBUTES.

         So now you can now more easily answer questions like, "Is there
         a dataset named 'foo' anywhere in the file?"

o UPS_CM_Sm_get_item, UPS_CM_Sm_malloc, UPS_UT_Mem_get_item
        (These routines deal with with getting/setting memory chunks)
        The argument for memory size had been a C long.  This translated
        to being a 4 byte int or an 8 byte int depending on compilation.
```

```
        Now, the argument is explicitly a long long (int8) removing
        ambiguity.

==========================
New UPS Version v-02-05-02
==========================

Machines Installed: theta, lambda, chi, bluemountain, q(s)

Please email ups-team@lanl.gov if you have any questions/comments.

Major Changes since v-02-05-00
==============================
o Tools
  lib/libuserd-HDF
  EnSight reader can now handle structured data where geometry data
  is defined along the axis (as opposed to each node having geometry).
  This is the coordinates_axis_[xyz] option below.

      If (coordinates):
        2d float dataset [n][3] - each row contains the x, y, z
        coordinates.
      If (coordinates_x, coordinates_y, coordinates_z):
        Each dataset is an nd dataset containing the corresponding
        component.
        Unlike global coordinates, the dimensionality of these
        datasets is important.  Each dataset must have the same
        dimensions - which will correspond to the "i,j,k" dimensions
        of EnSight...but in the opposite order.
        HDF dimensions of [2][3][4] will correspond to i=4,j=3,k=2.
        Usually, i/j/k corresponds to the number of nodes along the
        x/y/z axis respectively.
        When creating the values for the dataset, order along the
        x-axis, then y-axis, then z-axis.
        NOTE: for simple rectangular meshes, it is easier to use
        the coordinates_axis_[xzy] form below.
      If (coordinates_axis_x, coordinates_axis_y, coordinates_axis_z):
        Each dataset contains the values of that component along that
        axis.  The coordinates are created from all combinations of
        each component.
        For example, suppose you have 4 values along the x-axis, 3
        values along the y-axis, and 2 values along the z axis.
        the datasets coordinates_axis_[xyz] will have 4, 3, and 2
        values respectively.

o IO Package
  - optimizations:
    UPS_IO_Dataset_write() has always been a collective call (every
    process must call is).  Now, I tell HDF it is collective and
    HDF is able to optimize its operations.  This is used when
    UPS_AA_Opt_set() is called setting UPS_IO_OPT_ACCESS_WRITE to
```

```
    UPS_IO_ACCESS_ALL_PE.

  - new functions: UPS_IO_Ds_r_s, UPS_IO_Ds_w_s
    These functions simplify the reading/writing of datasets
    where each process has a single value.  These functions simply
    wrap more general UPS calls into 2 functions.

=========================
New UPS Version v-02-05-00
=========================

Machines Installed: nirvana, theta, lambda, chi, bluemountain, q

Please email ups-team@lanl.gov if you have any questions/comments
regarding the below changes.

Major Changes since v-02-04-00
==============================
o Tools
  bin/ups_io_rm

        Purpose:
        =======
        Remove an object (recursively) from an HDF file.
        Due to current HDF limitations, ups_io_rm will leave
        holes in the file.  To 'reclaim' this space, use ups_io_cp
        to recursively copy the old file to a new file.  This will
        remove any holes.

        Format:
          ups_io_rm <file> <object> [options]

        Options:
          -a <attribute>: remove only the specified attribute
          -f: do not error if object to remove does not exist
          -v: verbose output (screen output and ups_log.txt file
              created)
        Examples:
        ========
          - ups_io_rm foo.h5 /foo/pc/variables/per_element
            Remove the '/foo/pc/variables/per_element' object
            recursively in foo.h5.

          - ups_io_rm foo.h5 . -a EnSight_model -v
            Remove the 'EnSight_model' attribute from the root
            group in foo.h5 and be verbose about it.
        Notes
        =====
        - Due to current HDF limitations, ups_io_rm will leave
          holes in the file. To 'reclaim' this space, use ups_io_cp
          to recursively copy the old file to a new file.  This will
```

```
         remove any holes.

  bin/ups_io_diff.pl
       Added RMS to output.  I chose the "N-1" (sample) version:
              sqrt( sum((value_1 - value_2)**2) / (N-1) )

  lib/libuserd-HDF
       - Users may now set the environment variable LIBUSERD_HDF_VARIABLES.
         This variable contains a comma delimited string of variables to
         be used (all others will be ignored).

       - Structured meshes can now be read.  Any Part that has its own
         coordinates is assumed to be a structured part.  The dimension
         size of the coordinates datasets ("coordinates_x", "coordinates_y",
         "coordinates_z") specify the size of the structured part:
                coordinates_x[3][4][5] --> 5 i nodes,
                                           4 j nodes,
                                           3 k nodes
         In HDF, the last dimension is contiguous in memory.  For example,
         the position [1][2][3] is adjacent to [1][2][2] and [1][2][4].
         EnSight uses "neighbor" nodes to draw its structured mesh.
         So, the order of the coordinates is important if you want EnSight
         to draw a pretty picture.

         As an aside, one can have both structured parts (each part using
         its own nodes) and unstructured parts (each part using the
         same set of global nodes).  This allows one to have "particles"
         (and variables defined on those particles) by setting them as
         structured parts with their own nodes.

         See the README for more information.

       - This reader now can be used on Q machines (compaq, hp, whatever
         its called now) as well as SGI's.

o GS package - UPS_GS_START_INDEX added to UPS_GS_Get_item call:
          (int) The original start index for process 0.
          This value can be obtained if given the my_start_index
          argument (eg calling UPS_GS_Setup or UPS_GS_Setup_s_global).
          As with UPS_GS_GLOBAL_INDICES, an artificial value of -1 will
          be set if my_start_index was not given (UPS_GS_Setup_s_local).
          Note that the items UPS_GS_GLOBAL_INDICES and
          UPS_GS_INDICES_ACCESSED still return 0 based arrays.

o IO package - Major optimizations.

          Writing a dataset that is distributed in a cyclic manner across
          the processes can be very time consuming since each process is writing
          to various parts of the file at the same time.  I have added logic
          to write data in a "smart" way.  Master processes gather slave
          processes data to form contiguous chunks of the dataset.  They then
```

```
          write these contiguous chunks.

          The performance increase is substantial.  When using 8 processes
          and each process has a 100X100X100 chunk of a 100X100X800 integer
          dataset, there is a 20X speedup.  I expect that when using more
          processes or larger datasets, the speedup will be even greater.

          The default access for writing is UPS_IO_ACCESS_IO_PE (the IO pe
          does all the writing) which takes advantage of this data aggregation.
          UPS_IO_ACCESS_SERIAL (which was added) also takes advantage of
          this aggregation but has multiple master pes doing the writing.
          UPS_IO_ACCESS_ALL_PE has every process write its own data.
          NOTE: UPS_IO_ACCESS_IO_PE is a bad choice for poorly distributed data.

          The above options can be set by calling:
              UPS_AA_Opt_set( UPS_IO_OPT_ACCESS_WRITE, <access mode> );

o UT package
  - UPS_UT_Get_name_or_value() also takes UPS_UT_GET_COUNT as an argument.
    This returns the number of name/value pairs in the name_value struct.

  - UPS_UT_Sort_compress()
    Added function to sort and then compress (remove duplicate) an
    array.  The changes are done to the input buffer and an updated
    count is returned.  I needed this so I thought others might
    benefit from it.

==========================
New UPS Version v-02-04-00
==========================

Machines Installed: nirvana, theta, lambda, chi, bluemountain
Directory: /usr/projects/ups/latest -> v-02-04-00

Please email ups-team@lanl.gov if you have any questions/comments
regarding the below changes.

Major Changes since v-02-03-01
==============================
o Tools
  Tools built on top of UPS are now being written and distributed with UPS.
  Documentation for these tools (and all UPS documentation) can be found
  in doc/<tool>.

  lib/libuserd-HDF.so
       Purpose: create an EnSight reader that can read HDF files

       For information, please see doc/files/README.libuserd-HDF.
       The file example_mesh.h5 is an example mesh that can be read
       with the reader.
```

```
     Included in the documentation directory is the reader code
     libuserd-HDF.c and test_libuserd-HDF.c which give examples of how
     to write and read a mesh.

bin/ups_io_cp
     Purpose: copy object from one hdf file to another hdf file

     Format:
       ups_io_cp <from file> <from object> <to file> <to object> [options]

     Options:
       -R: recursive copy
       -kfrom <sort dataset>: sorting key dataset of from dataset
       -kto <sort dataset>:   sorting key dataset of to dataset
           if only one sort key dataset is specified, use the
           same sort key dataset name for both.
       -v: verbose output (screen output and ups_log.txt file
           created).

     Example:
       - copy from file correct.h5 the group     RHO
              to file   foo.00000.h5 renaming it RHO_correct
              while sorting on dataset node_ids.
              The copy is recursive and verbose.

           ups_io_cp correct.h5   /foo/pc/variables/per_node/RHO
                     foo.00000.h5 /foo/pc/variables/per_node/RHO_correct
                     -kfrom       /foo/pc/geometry/node_ids -R -v

bin/ups_io_diff.pl
     Purpose: print differences between a dataset existing in two files.

     Usage
     =====
       ups_io_diff.pl <file 1> <file 2> <datset>
                      [-l]
                      [-or]
                      [-k <sort key dataset>]
                      [-a <absolute tolerance>]
                      [-r <relative tolerance>]

       -l
          long output
          Print information about each of the differences.

         -k <sort key dataset>
            Values of the datasets are to be compared by the key given in
            this dataset.  The sort key datasets for each file are checked
            to make sure all keys in 1 dataset exist in the other dataset.

            If the sort key dataset has a different number of items than
```

```
            the dataset, each sort key item will correspond to a consecutive
            group of values of the dataset.  For example, you can have a sort
            key of the dataset "node_ids" with 10 items in it (10 nodes) and
            the dataset you are diffing is "foo" which is a "10X3" dataset
            (1 value per node with X, Y, and Z components).

         -a <absolute tolerance>, -r <relative tolerance>
            Values are said to be "the same" if the absolute/relative
            difference is less than or equal to the absolute/relative
            tolernace.

            Both -a and -r flags can be specified at once.
            If both -a and -r flags are specified,
            a value is "the same" if it is within (less than or equal to)
            _both_ the absolute _and_ relative tolerances.
            See the "-or" flag below.

              Absolute Difference{[0,)}  = abs(value1-value2)
              Relative Difference{[0,1]} = abs(value1-value2)/
                                           (abs(value1)+abs(value2))

         -or
            "or"
            This flag changes the default behavior when given both absolute
            and relative tolerances. With this flag,
            a value is "the same" if it is within (less than or equal to)
            _either_ the absolute _or_ relative tolerances.

      Return Value
      ============
       If a difference is found, information is sent to the screen and
       a -1 is returned.
       If no differences, no output and 0 is returned.

      Examples
      ========
       1) Print difference info (short format) between 2 datasets:
          ups_io_diff.pl example_mesh1.h5 example_mesh2.h5
             "/EnSight_model/variables/per_node/UPS:N:scalar 0/unstructured"

       2) Print difference info (long format) where the
            absolute tolerance is 1e-8 _or_ relative toleralce is .1
                (values same if within either absolute or rel tolerances)
                (remove "-or" if you want values to be same if within both)
            while keying on a sort key dataset
            between 2 datasets:
          ups_io_diff.pl example_mesh1.h5 example_mesh2.h5
             "/EnSight_model/variables/per_node/UPS:N:vector 0/unstructured"
             -l -k /EnSight_model/geometry/node_ids -a 1e-8 -r .1 -or

o UPS_DT_BOOL datatype:
```

```
  The datatype UPS_DT_BOOL has been removed....sort of.
  I has just been setting this datatype to the same
  as UPS_DT_INT4.  When dealing with Fortran logicals, this
  is true.  However, C++ booleans are 4 bytes, but 1 byte.
  So, this datatype is ambiguous/confusing.  So, I have
  removed it.

  For those of you that were using UPS_DT_BOOL, replace
  it with UPS_DT_INT4/UPS_DT_INT.

  For machines I have been on, use UPS_DT_CHAR when dealing
  with C++ booleans and use UPS_DT_INT4 when dealing
  with Fortran logicals.

o Environment variables that affect UPS
  UPS environment variables are now listed in 1 place - ups.h.
  One can search for the work 'environment' to find the enums that
  list them.

    - Environment variables supersede function calls made by the user.
      This allows one to change behavior without needing to recompile
      code.
    - These variables are read during UPS_AA_Init() and are not reread.
      So, modifying them during a run has no affect.

o Run Mode: parallel or serial
  One can set the "run mode" for UPS runs.
  Currently, this "run mode" dictates whether single pe runs should
  still go through MPI or call the UPS serial code.  See
  UPS_AA_RUN_MODE_SERIAL for an explanation of how this could
  possibly be useful.

  Use UPS_AA_Opt_get/UPS_AA_Opt_set with UPS_AA_OPT_RUN_MODE.
  These may only be set before calling UPS_AA_Init()
  You may set the following as environmental variables as well:
  (in mutually exclusive groups)
     - setenv UPS_AA_RUN_MODE_PARALLEL
       setenv UPS_AA_RUN_MODE_SERIAL
  Remember that all environment variables must be propagated to
  all the processes.

    UPS_AA_RUN_MODE_PARALLEL=0, /* Default (must equal 0 to match init)
                                 * This option causes the underlying
                                 * communication protocol (eg MPI) to
                                 * be called regardless of the number of
                                 * processors being used. */
    UPS_AA_RUN_MODE_SERIAL /* This option causes serial code (single
                            * process) to be called instead of the
                            * underlying communication protocol (eg MPI).
                            * UPS has the same functionality when running
                            * serial as it does running parallel with
```

```
                                * one process.
                                * This option was added mainly for running
                                * with the HDF-EnSight reader which uses
                                * UPS IO routines but cannot run under MPI.
                                * There are other tools that also have this
                                * restriction. */
```

o Guides
  The pdf flavors of the guides (UserGuide.pdf and DeveloperGuide.pdf)
  now are hyperlinked....sort of. For some reason, some of the hyperlinks
  miss the page they are pointing to by several pages. But, its close.
  The hyperlinks will still list a page number so just page down several
  pages to get to the correct page.

o Those building UPS from source - out of tree builds
  To build out of tree (source in 1 location, binary files in another),
  set the following environment variables:
      UPS_PREFIX = absolute path to output directory
      CONFIGURE_PL_DIROUT = set to UPS_PREFIX
  Without setting these, the default is to build as before - in tree.
  For more information, run 'configure.pl help'. One can also get other
  info by running configure.pl without any arguments.

o UPS IO Package:
  The user may now set their own file_id/group_id values.
  The default is still to let UPS pick them for you.
  This allows a more fortran-like approach.

  This option is set with a call to UPS_AA_Opt_set() with
  UPS_IO_OPT_LOC_ID_USERS set to UPS_DT_TRUE.

  Users may now get the id used by the underlying protocol
  with a call to UPS_IO_Loc_item_get with options
  UPS_IO_LOC_ID_PROTOCOL and the loc_id obtained from a
  UPS_IO_(File|Group)_open call (eg get the HDF file_id
  given a UPS file_id).

o UPS IO Package: UPS_IO_INFO_DATATYPE_SIZE
  UPS_IO_INFO_DATATYPE_SIZE was ill-defined for datatype of
  UPS_DT_STRING. When getting information about a dataset or
  attribute, one can call UPS_IO_Info_item_get with an argument
  of UPS_IO_INFO_DATATYPE_SIZE to get the number of bytes in the
  datatype. Previously, for UPS_DT_STRING, UPS_IO_INFO_DATATYPE_SIZE
  would return the number of chars (same as UPS_IO_INFO_NITEMS).
  Now, UPS_IO_INFO_DATATYPE_SIZE for UPS_DT_STRING will return the
  number of bytes in a char (1).

  To get the number of elements in a dataset or
  attribute, use UPS_IO_INFO_NITEMS (local to the pe) or
  UPS_IO_INFO_NITEMS_TOTAL (number of elements total). The
  functionality of UPS_IO_INFO_NITEMS/NITEMS_TOTAL has not changed.

```
o UPS IO Package: new open method
  Added a new open method: UPS_IO_OPEN_READ_WRITE.  Formerly,
  there had only been UPS_IO_OPEN_CREATE (write a new file)
  and UPS_IO_OPEN_READ (read a file).  UPS_IO_OPEN_READ_WRITE
  will open an existing file for read/write ability.

  Checks are now made to make sure you are not doing things like
  trying to write to a read-only file.

o UPS IO Package: UPS_IO_Info_create_self, UPS_IO_Rm
  Want to get an info_id about a single object without having to
  to through
     UPS_IO_Filter_set
     UPS_IO_Info_count
     UPS_IO_Info_create
     UPS_IO_Filter_set

  Well, now you can!  UPS_IO_Info_create_self can be used to
  return a count (0 if object not found and 1 if it is) _and_
  the info_id.  This shortcut made my life bearable and it just
  might make yours as well...

  Also, With UPS_IO_Rm, there is now a way to remove specific objects
  from a file.

o UPS IO Package: UPS_IO_Info_item_set with UPS_IO_INFO_DATA_TOTAL
  When called with UPS_IO_Info_item_set, this modifies the info_id
  obtained from a info create call so that the entire dataset will
  be read in
  This is a short curcuit of the following calls (that I was making
  a thousand times and was starting to get annoying):
          1- UPS_IO_Info_item_get: UPS_IO_INFO_DIMS_TOTAL
          2- UPS_IO_Info_item_set: UPS_IO_INFO_DIMS
          3- UPS_IO_Info_item_set: UPS_IO_INFO_STARTS
  Remember, the default behavior is for each pe to read in only what it
  wrote out.  Use this option to UPS_IO_Info_item_set to read in the
  whole dataset.

o UPS IO Package - file type
  One can now get the file type (UPS_IO_PROTOCOL_HDF,
  UPS_IO_PROTOCOL_UDM, ...) from the name of the file (the function
  UPS_IO_File_type()) or a location id (the function
  UPS_IO_Loc_item_get() ).  During read calls, UPS will detect the type
  of the file and read it correctly.

o UPS IO Package - UPS_IO_Attr_write
  Now, all processes must have the same arguments.  Previously, only
  the data on the io_pe was relevant.  This was causing confusion
  as to what arguments needed to be the same and which arguments could
  be different.
```

```
o UPS UT Package: UPS_UT_Checksum_get
  Supply a buffer and a count...get back a "checksum" quantity.
  Currently, only 64 bit CRC is supported.  In the future, others
  can be added.

o UT Package - UPS_UT_Dt_change()
  One may now call this routine to change an input buffer of one
  datatype to an output buffer of another datatype.
  Only simple conversions (integer <--> float) are supported.
  C assignment is done (eg float to integer is truncated and
  overflow error might occur).  So, one must use this carefully.

  This is used in IO packages reading of attributes and datasets
  if the read datatype does not match the file datatype.

==========================
New UPS Version v-02-03-01
==========================

Machines Installed: nirvana, theta, lambda, chi, bluemountain
Directory: /usr/projects/ups/latest -> v-02-03-01

Please email ups-team@lanl.gov if you have any questions/comments
regarding the below changes.

Major Changes since v-02-03-00
==============================
1) UPS_CM_Context_free:
   This function has been added to allow users to free UPS resources
   when done with a context set by UPS_CM_Set_context.  As most users
   do not have more than a few contexts, this call is not necessary.
   However, we provide it for those users who wish to be especially
   tidy.

2) UPS_IO_Attr_write_s:
   One can now write an attribute specifying the size (ndims/dims) as
   arguments.  Previously, only single element attributes were
   supported (eg 1 double).  Now, you can write multi-dimensional
   attributes using this new function.

3) UPS_IO_Info_item_get:
   The datatype of UPS_IO_INFO_NAME_LENGTH (the length of a name) has
   been changed from an int4 to an int8 to better match other
   variables.

4) UPS_IO_Loc_item_get, UPS_IO_Loc_item_set:
   These functions have been added to allow users to get/set
   information related to a loc_id obtained from a call
   to UPS_IO_File_open or UPS_IO_Group_open.
```

```
   This is similar to what can be done already with
   UPS_IO_Info_item_get/set and info_ids obtained from
   a call to UPS_IO_Info_create.

5) UPS_IO_Filter_get, UPS_IO_Filter_set:
   One may now set a filter to the UPS_IO_Info_count/UPS_IO_Info_create
   calls.  The count/create calls are used, for example, to
        count: tell me how many members a group has
        create: give me the array of info ids for those members.
   Previously, to see if a member existed, one would have to do the
   following:
        1) UPS_IO_Info_count --> n matches
        2) create array of length n
        3) UPS_IO_Info_create --> ids array of length n
        3) for i = 1 to n
        3.1) get name of ids[i]
        3.2) if name matches, the member exists

   Now, one can:
        1) UPS_IO_Filter_set with name of member in question
        2) UPS_IO_Info_count will return 0 (not found) or 1 (found)

   To turn off the filter, call UPS_IO_Filter_set with a 0 length
   string.

6) IO Datasets:
   We now support up to 10 dimensional datasets (formerly had only supported
   2-d).  If you want more, just tell me.

   Remember, the order of the data buffer to be written is along
   the last dimension first (which is how it is layed out in memory).

   Upon writing, a process now has the following ways to specify what
   portion of the dataset it will write:

   a) Default:
         All dimensions except the first dimension (slowest
         moving dimension in memory) are the same for every
         process.  The global dataset is written in processor
         order.
   b) UPS_IO_INFO_STARTS:
         Each process specifies where its starting position in the
         global array.
   c) UPS_IO_INFO_PGRID_DIMS (and optionally UPS_IO_INFO_PGRID_ORDER)
         Each process specifies how many processes are along each
         dimension of the dataset.  The default ordering is along
         the last dimension first (just as the ordering of the data
         in the dataset).  This may be modified by specifying the
         processor order with UPS_IO_INFO_PGRID_ORDER.

7) IO Package Sequential File Access:
```

```
   UPS-IO can now provide sequential-like file access.  The goal
   is to have PSFlib-like functionality.

   Essentially we choose the names of datasets so you do not
   have to.  When calling UPS_IO_Dataset_write without a name argument,
   a name is created with an internal count as part of the name.
   This count is then incremented in preparation of the next write.
   The same is done for reading.

   Note, we encourage users to instead provide their own
   names/structure to the file so that the file is more self
   descriptive.

   Please see UPS_IO_Dataset_read in the UserGuide
   for example/discussion.

8) Write using n pes but read using m pes (m != n):
   Previous behavior was to read in the whole dataset automatically if
   the number of processes that wrote the dataset did not equal the
   number of processes reading the dataset.  This too easily lead to
   user error.  Now, if the number of processes do not match, you
   must set UPS_IO_INFO_DIMS and UPS_IO_INFO_STARTS.

9) Serial UPS:
   A pure serial version of UPS is now available.  This version will
   have the same functionality as running your code with 1 MPI process
   except that MPI is not used at all.  This might be useful when using
   tools (eg insight) that do not work well with MPI.

   Currently, this option of UPS will not be installed unless requested.


==========================
New UPS Version v-02-03-00
==========================

Machines Installed: nirvana, theta, lambda, chi, bluemountain
Directory: /usr/projects/ups/latest -> v-02-03-00

Major Changes since v-02-02-11
==============================
1) New IO package added:
   UPS offers IO functionality based on Hierarchical Data Format
   (HDF http://hdf.ncsa.uiuc.edu/HDF5).  A file written with
   HDF has a structure similar to a Unix directory structure (eg
   HDF datasets/groups are similar to Unix files/directories
   respectively).

   UPS has functions written on top of HDF that simplify access
   to the file.
```

```
   The file objects are:

     o- groups - like Unix directories
     o- datasets - like Unix files (arrays)

        Currently, UPS has simplified the reading/writing of datasets
        (arrays) by assuming that processes own consecutive contiguous
        chunks of the dataset.  All each process needs to specify is
        the size of its chunk.
        Please see UPS_IO_Dataset_read in the UserGuide for
        example/discussion.

     o- attributes - simple datasets attached to groups or datasets

        These can be used to describe the object to which they are attached.

   One can use the IO package functions to create self-descriptive
   data files.  One can also examine these files with command line
   tools h5ls and h5dump which are installed in the UPS bin directory.

   In the near(?) future, the UPS IO package will layer on top of
   other packages (UDM - Unified Data Model) which will give
   users access to greater functionality.

   Note: The user must now link in libhdf5.a (which is located in the
         UPS lib directory).

   The IO reference section in the UserGuide
   describes each function and has exquisite examples.

2) UPS_DP_Combiner functionality change:
   The (max|min)loc operation had returned a location relative
   to the masked array.  Now, it returns a loc relative to the
   entire array.  User requested to do it this way.  And it makes
   sense to me that it would be more useful.

3) ups.h include statements and C interface routines:
   A while back, I had removed the "include <mpi.h>" line from
   master_ups.h (remember, master_ups.h gets converted to ups.h and
   that file is included by users).  However, the following functions
   had arguments that were of MPI types:
       UPS_CM_Get_context:  MPI_Comm
       UPS_CM_Set_context:  MPI_Comm
       UPS_CM_P_group_item: MPI_Comm
       UPS_CM_Csend:        MPI_Request
       UPS_CM_Cwait:        MPI_Request

   Now, the actual argument were hidden by typedefs that were assigned
   differently via #define code.  ie:
       typedef MPI_Comm UPS_CM_Context
```

```
   However, this required putting a "#include <mpi.h>" inside
   master_ups.h and thus it would be visible to users including
   ups.h.

   I have not made a change so that the arguments of the functions
   that were mpi datatypes are now pointers to void.  This means
   we now no longer have to have "#include <mpi.h>" in ups.h

   So, what does this mean to the user interface?  Well, very little
   probably:
           Fortran: no change
           C: pass the address instead of the variable in question
              UPS_CM_Set_context( comm ) --> UPS_CM_Set_context( &comm );

   C Users using the typedefs UPS_CM_Context/UPS_CM_Request can now
   use the #defines UPS_DT_PROTOCOL_COMM_TYPE and
   UPS_DT_PROTOCOL_REQUEST_TYPE.  If they really do not want to know
   the type of the argument they get from UPS_CM_Get_context, they
   can call UPS_DT_Sizeof and get the size in bytes and just malloc
   a chunk (mentioned in the comment headers of the affected
   routines).

   Fortran users are not affected by this because they have always
   passed in int* to these routines.  As a check, upsp_dt_init
   now checks to assure that the size of an int equals the
   actual size of the C flavor.

   For the developer, since you are receiving a void*, you will
   have to cast it before use: (UPS_DT_Protocol_comm*)comm.

4) UPS_UT_Get_name_or_value new function:
   Added the function UPS_UT_Get_name_or_value.  I had created this
   when doing io work for another code team...and I thought it might be
   useful in general.  The user creates an array of
   UPS_UT_Name_value_struct's and passes that in with either a name or a
   value.  The function returns the value or name (respectively).

5) Move get/set io_pe routines from gs to aa component:
     UPS_GS_Get_io_pe --> UPS_AA_Io_pe_get
     UPS_GS_Set_io_pe --> UPS_AA_Io_pe_set
   Formerly, the io_pe was used only by the GS package.  Now, it is
   also used by the IO package and in the future, might be used by
   other packages.  That is why I decided to move it to the AA (general)
   package.

6) UPS_AA_Opt_get/UPS_AA_Opt_set functions added
   These functions are used to get/set options that dictate how UPS
   behaves during run-time.  This will be where the user may set things
   like buffer sizes, message aggregation techniques, ... .
   Take a look at the UserGuide for more information on the different
   options available.
```

```
7) New links in install directory for easier building
   Currently, when linking to an installed UPS, the architecture dependent
   directories (bin, include, lib) are located at:
        latest/lib/SGI64_mpi/libups.a
        latest/include/SGI64_mpi/ups.h
   In other words, lib/<ARCH>.

   I have added links so that you may now access architecture dependent
   files by:
        latest/lib/SGI64_mpi/libups.a
        latest/include/SGI64_mpi/ups.h
      -and-
        latest/SGI64_mpi/lib/libups.a
        latest/SGI64_mpi/include/ups.h

   In other words, both lib/<ARCH> and <ARCH>/lib are now available via
   soft links.
```

4 Directory structure

Below is a chart of the directory structure...followed by a listing of file name conventions and meanings.

Directories under UPS (**directory**|**subdirectory**|**sub-subdirectory**|**...**)			
Makefile			
aux	hdf *other auxiliary directories*		
bin	*Architecture directories*	h5ls	
configure.dat configure_local.dat configure.pl			
doc	DeveloperGuide		
	UserGuide	header2latex.pl	
	other Docs		
include	UPS.F UPS_CONST_MOD.F		
	Architecture directories	ups.h upsf.h upsf77.h *other user includes*	
	master_ups.h sync_f_include.pl		
lib	*Architecture directories*	libups.a	
make.inc			
peer_review	lmdm	ToDo.lmdm	
	other developers peer review directories		
script	Parse_file.pm ups_aa_statistics_plot.pl *other generic scripts*		
src	aa	ups_aa_init.c ups_aa_terminate.c	
	gs	UPSF_GS_MOD.F UPSF_GS_SETUP.c UPS_GS_SETUP.c ups_gs_setup.c ups_gs_setup_f.F ups_gs_setup_f77.c *other user API files* upsp_gs.h	
		utils	upsp_gs_init.c upsp_gs_terminate.c *utility files not seen by user*
	other UPS packages like aa, cm, dp, ...		
testing	gs	fortran	*source for fortran gs tests*
		other tests for the gs package	
	other tests for UPS packages like aa, cm, dp, ...		
tools	EnSightReaders	*source for this tool*	
	other tools		

file name conventions/descriptions	
name	*meaning*
[wW].pl	perl script (configure.pl)
[wW].tex	latex files (DeveloperGuide.tex)
(2l)	directory pertaining to a package (cm)
(2l)i_[w]	directory containing internal package source (cmi_mpi)
aux	directory containing collaborator code
configure.dat	top configure.dat = $(UPS_PREFIX)/make.inc
configure_local.dat	local configure_local.dat = $(DIR_DSTDIR)/make_local.inc
configure.pl	perl script processing configure.dat creating the make.inc
doc	directory containing userguides, conference reports, ...
include	location of user include files
internals	contains architecture dependent directories for UPS packages
header2latex.pl	creates latex reference pages from headers in ups_(2l)_[w].c files
libups.a	user UPS library
make.inc	Makefile include file generated by configure.pl
Makefile	recursively builds whatever is needed at the current directory
master_ups.h	master C include file used to generate user include files
Parse_file.pm	Parses various file types and creates variables
script	contains scripts that are or have been used
src	directory containing UPS source
testing	directory containing tests for the different UPS packages/APIs
UPS.F	Used to generate UPS.mod (for fortran USE statement)
UPS.mod	for fortran USE statement
UPSF_(2L)_MOD.c	Contains names of the fortran modules
UPSF_(2L)_[W].c	Fortran API created manually (UPSF_CM_ALLREDUCE.c)
UPS_(2L)_[W].c	F77 API created manually (UPS_CM_ALLREDUCE.c)
UPS_CONST_MOD.F	creates constant module USE'd by UPS.F
ups_(2l)_[w].c	C API (ups_cm_send.c)
ups_(2l)_[w]_f.F	Fortran API created automatically (ups_cm_send_f.F)
ups_(2l)_[w]_f77.c	F77 API created automatically (ups_cm_send_f77.c)
ups_aa_statistics_plot.pl	Perl script that creates ps plots from ups_log.txt output
ups.h	user include file generated from master_ups.h include file
upsf.h	fortran version of the ups.h (generated from ups.h)
upsf77.h	f77 version of the ups.h (generated from ups.h)
upsp_(2l).h	private include for API (upsp_cm.h)
utils	directory containing utility files

key	
expression	*meaning*
[]	at least one occurrence
()	single occurrence
(l)	lower case letter
(L)	upper case letter
(2l)	2 lower case letters
(2L)	2 upper case letters
[w]	lower case letters, underscores, and/or numbers
[W]	upper case letters, underscores, and/or numbers
[wW]	lower/upper case letters, underscores, and/or numbers

Listed below are more in depth explanations of directories/files.

4.1 Main UPS Directory

The two important text files in this directory are `Makefile` and `configure.dat`.

- `Makefile`

  All major operations can be done from this level.

  The most used options are:

  - `gmake`
    This builds the libraries, tests, and User/Developer Guides. Also, the tests will run and you will be notified of their success/failure.
  - `gmake clean`
    This cleans all .o files.
  - `gmake clean_compile`
    This cleans and also removes all libraries.
  - `gmake install_dev`
    This installs the ups product for the general public.
  - `gmake help`
    Prints help message.

- `configure.dat`

  This input file is used by configure.pl to create make.inc (which is included in all makefiles). To change the options, edit this file and change the values defined by "CONFIGURE_OPTIONS". To add new variables, simply add them to this file.

4.2 aux

This directory contains 3rd party software used by UPS. Some examples of this are:

- `hdf`

  Hierarchical Data Format (file io product)

These libraries are built and either put directly into libups.a or kept in a separate library that users explicitly link to when compiling their code.

4.3 doc

UPS documents are kept here. This includes UserGuide, DeveloperGuide, and conference papers (like for SC99 and HPCU99).

4.4 include

The purpose of the include directory is to create the different include files needed by different languages (specifically C, C++, Fortran, and Fortran-77).

At the top level, there is `master_ups.h` from which all the other ".h" includes are generated. This file contains enumerated types and prototypes to the user interface files written in C. Also, at this level, there are the files `UPS.F` and `UPS_CONST_MOD.F` which create `UPS.mod`.

All of the files (including the fortran source) are put here because they have the same purpose...create user include files.

The "include" files are placed in subdirectories whose names represent different compile time options.

- `SGI64_mpi`

  compiled on the SGI, with 64 bit addressing.

4.5 lib

The libraries users need to use UPS are placed in subdirectories whose names represent different compile time options.

- `SGI64_mpi`

  compiled on the SGI, with 64 bit addressing.

4.6 peer_review

This directory contains directories for each developer. Under those directories, developers maintain their own personal ToDo lists. These lists serve as reminders for what the developer is doing/wants to do in the future and serve to allow other team members to see what everyone is doing.

4.7 script

This directory contains scripts that have been used or are used. One important file in this directory is `Parse_file.pm`. This perl module parses various file types and creates variables. header2latex.pl, create_fortran_interface.pl, create_f77_interface.pl, sync_f_include.pl, and precommit.pl use this module.

Another script (which is copied over during an install to allow user access) is ups_aa_statistics_plot.pl. This scripts takes the output file ups_log.ps and uses gnuplot to create postscript graphs.

4.8 src

The src directory contains the main part of UPS. This includes user interfaces (`src/cm`) and utility code (`src/cm/utils`).

The user interface layer (eg `src/cm`) contains all the subroutines to which the user has access. These routines contain code that is in general invariant on the architecture (eg SGI vs. Intel) or the underlying communication (eg MPI or PVM). Minor differences may be #ifdef'd in these files. Major differences (eg a new solver package) might require the creation of a new internals or utils directory.

The user interface directory maintains consistent interfaces so that UPS may change underlying protocols (eg MPI to PVM communication protocols) that are effectively invisible to the user.

For example, in the CM package we have pulled out all the MPI functions that are currently used and placed them inside "core" routines:

```
UPS_CM_Send()--calls-> utils/upsp_cm_core_send()--calls-> MPI_Send()
                                                 |--calls-> pvm_send()
                                                 |--calls-> serial op
```

When we switch from using MPI to PVM or serial, we only need to change the "core" functions via #ifdefs. This allows minimal code changes for developers and no code modifications for users.

The following is an example of how an MPI internals directory might be built for the CM package (as opposed to the current "core" method for insulating MPI calls).

```
UPS_CM_Send()--calls-> internals/cmi_mpi/upsi_cm_send()
             |--calls-> internals/cmi_pvm/upsi_cm_send()
             |--calls-> internals/cmi_serial/upsi_cm_send()
```

There might be several different internals (eg `cmi_pvm` could denote a PVM version). Note that there is a corresponding user interface "ups" file for every internal "upsi" file. The internals directory contains protocol-specific versions of user interface files.

The correct protocol directory (cmi_mpi/cmi_pvm/cmi_serial) is compiled via the make.inc Makefile variable CM_INTERNAL.

It is up to the developer to decide whether the code modifications for different protocols should be handled in #ifdefs, core functions, or new internals. In general, try to minimize the amount of code duplication so that changes do not have to be propagated across multiple files.

- Minor Changes –> #ifdef's directly in user interface

  UPS_GS_Distribute():

  The tflop implementation of MPI_Scatterv is broken in that all processes must have the counts array instead of just the root pe. So, when on the tflop, #ifdef'd code bcast's the counts array first.

- Moderate Changes –> "core" utils functions

  UPS_CM_Send() using the "core" method:

  We use MPI calls in many places...however there is little difference between most of the MPI calls we use and their PVM counterparts. So, we insulate the MPI/PVM/serial flavors in core routines that are then called by UPS functions.

- Massive Changes –> new internals directory

  UPS_CM_Send() using the "internals" method:

  When the functionality is vastly different (or the resulting #ifdef'd code is too complex to follow, a new internal should be created.

4.8.1 Automatic Generation of Interfaces

Where possible (eg easily automated), the Fortran and F77 interfaces are automatically generated from the C interface comment header. The C comment header has several fields that specify what is needed in the Fortran interface. See the beginning of the Reference section of the UserGuide for details on what those fields must look like.

The script routines create_fortran_interface.c and create_f77_interface.c are called as suffix rules in make.inc to convert .c to _f77.c and _f.F files. The Makefile in the src/<package> contains the variables:

- COBJS_PROC - F77 autogenerated interface files
- FOBJS_PROC - Fortran autogenerated interface files

Interfaces that cannot be autogenerated (eg some preprocessing of input data must be done), should be listed under COBJS or FOBJS respectively.

- UPS_CM_ALLREDUCE.o - F77 interface file manually created
- UPSF_CM_ALLREDUCE.o - Fortran interface file manually created

4.8.2 src User API

The following files are required for the aa package User API level:

User API Files: AA Package Only: src/aa	
Example File Name	**Meaning**
`ups_aa_init.c`	User C/C++ interface file (initialization routine). `Accessed by` user/developer functions.
`ups_aa_terminate.c`	User C/C++ interface file (termination routine). `Accessed by` user/developer functions.
`ups_aa_<function>.c`	User C/C++ interface (other routines). `Accessed by` user/developer functions.
`UPS_AA_<FUNCTION>.c`	User F77 interface that could not be auto generated. `Accessed by` user/developer functions.
`UPSF_AA_<FUNCTION>.c`	User Fortran interface that could not be auto generated. `Accessed by` user/developer functions.

The initialization and termination routines do the following:

- aa initialization routine - initialize all packages
- aa termination routine - terminate all packages

Only the aa package will have user-visible initialization/termination routines. Other packages will have their init/terminate routines in their utils directory (eg src/cm/utils/upsp_cm_init.c).

The following files are required for other packages in general (cm used as an example):

User API Files: All Packages: src/cm example	
Example File Name	**Meaning**
`upsp_cm.h`	Private include visible to all UPS routines. `Accessed by` developer functions. Contains prototypes and general structures. See below.
`ups_cm_<function>.c`	User C/C++ interface (other routines). `Accessed by` user/developer functions.
`UPS_CM_<FUNCTION>.c`	User F77 interface that could not be auto generated. `Accessed by` user/developer functions.
`UPSF_CM_<FUNCTION>.c`	User Fortran interface that could not be auto generated. `Accessed by` user/developer functions.

The include file `upsp_cm.h` defines the struct `upsp_cm` that contains variables that can be accessed by any UPS routine. This structure must contain information about whether this package has been initialized and which packages this package initialized. Other information may be included.

```
typedef struct {
  int
    host_numpes,              /* number of pe's per host        */
    host_penum,               /* penumber wrt host              */
    initialized,              /* if this package is initialized */
    num_comms,                /* number of communicators        */
    numhosts,                 /* number of hosts                */
    numpes,                   /* number of pes                  */
    penum;                    /* penumber                       */
```

```
  UPS_CM_Comm   *comms;        /* communicators                  */

} ups_cm_globals;
UPS_EXTERN ups_cm_globals upsp_cm;
```

Some routines in the internals directory will be expected to modify these variables.

The include file `upsp_cm.h` also includes the internals include file `upsi_cm.h` (located in the internals directory).

4.8.3 src internals

Only files that are dependent upon certain protocols (eg Architecture or underlying communication methods) should be found in this layer. That is, some user interface files contain only information that is protocol invariant. In this case, the user interface file does all the work and an internal file is not needed.

The following files are required in each package's internals directory (`src/gs/internals/gsi_mpi` is used as an example):

User API Files: gs/internals/gsi_mpi example	
Example File Name	**Meaning**
`upsi_gs.h`	Internal include file included by upsp_gs.h (which is located in src/gs) `Accessed by` internals functions.
`upsi_gs_distribute.c`	File that has MPI calls. `Accessed by` user C interface functions.
`utils/upspi_gs_setup_generic.c`	Private internal with no corresponding user interface. `Accessed by` internals/user interface functions.

The include file `upsi_gs.h` may contain information that is accessible to other UPS routines. The variables should be in an equivalent form as the struct described in `upsp_cm` (page 45) above except the struct should have the name, for example, `upspi_cm` (which represents that it is a private internal).

Even if the internal has no information it wishes to share, the include file must exist. This is so that upsp_gs.h can automatically include upsi_gs.h which might be needed for other internals (eg gsi_pvm).

All files include `ups.h` and `ups_aa.h` in order to include all global variables accessible to UPS routines (in order to have access or modify these variables).

Internals directories are named (with `cm` as an example):

- `cmi_mpi` (MPI communication protocol)
- `cmi_pvm` (PVM communication protocol)
- `cmi_serial` (Serial code).

4.8.4 src utils

As mentioned earlier in the "src User API" section, the aa package has user accessible init/terminate routines. These aa routines init/terminate all the packages. Other packages have init/terminate routines in their prospective utils directories (eg src/cm/utils/upsp_cm_init.c) which are called by the init/terminate routines in the aa package.

The utils directory contains files that can be called from other packages but are not meant to be called directly by the user.

4.9 testing

The testing directory contains code that tests all of UPS. This includes testing all functions for correct functionality, stressing functions, and simulating incorrect use.

Each packages has its own subdirectory under testing. Under that level, subdirectories named after specific types of usage are found. The directories containing tests are specifiec in configure.dat,

Although the tester may print out a variety of information (eg timings), the final output must assign a pass/fail rating in the following format.

```
================================================
**Passed** c xtest_cm passed
================================================
```

This format allows for automatic post-process of the output from all package testing.

It is preferable to keep output messages sent to the screen to a minimum. This way, errror messages are easily seen. For example, timing information should be sent to an output file.

4.10 tools

The tools directory contains tools built upon UPS. For example, `EnSightReaders` is a user defined EnSight reader built upon UPS IO. In general, the source for each tool is small. The source, documentation, and tests are all in one directory.

When adding a tool, just mimic what is done in the other tools.

5 Programming Practices

Sharing consistent programming practices between developers saves time. UPS source is written in a style such that any developer can add code in a way that is similar to existing code. The following guidelines are followed.

5.1 Variable Name conventions

Just like the names of files and directories, variables follow a convention.

Variable Name Conventions[3]		
type	**convention**	**example**
General Names		
Public	ups	`ups_cm_x`
Private	upsp	`upsp_cm_x`
Internal	upsi	`upsi_cm_x`
Private Internal	upspi	`upspi_cm_x`
Makefile		
Variable definitions	[W]	`CFLAGS`
Non-existent targets	[w]	`all`
C/Perl Source (F77 API written in C)		
Variable definitions	[w]	`num_flags`
C Subroutine API Names	UPS_(2L)_(*L*)[w]	`UPS_CM_Send`
F77 Subroutine API Names	#define'd to be lower case and have correct underscores	`UPS_CM_SEND =` `ups_cm_send_`
C Subroutine Internal Names	upsi_(2l)_[w]	`upsi_cm_send`
User Enumerated Definitions	UPS_(L)[w]_enum	`UPS_Error_enum`
User Enumerated Values	UPS_[W]	`UPS_ERROR_AA_INIT`
Package Private Data Structure	upsp_(2l)	`upsp_cm`
Fortran Source (F77 API written in C - see above)		
Variable Definitions	[w]	`num_flags`
Fortran Subroutine API Names	UPSF_(2L)_[W]	`UPSF_CM_SEND`
User Parameter Variables	UPS_[W]	`UPS_ERROR_AA_INIT`
Preprocessor		
Variable definitions	[W]	`MAX_TESTS`
#ifdef Variables	[W]	`USE_HBP`

5.2 Style

Style represents the general look of source. Whitespace is important for code readability (and to allow for comments). Comments allow for other developers (or even the original coder) to look at the code and quickly understand its purpose.

`Xemacs` [5] and the GNU Coding Standards [4] provide the basis for formatting.

Discussed below are spacing and comments.

[3] See the key in Directory Structure (section 4) for an explanation of the metacharacters.

5.2.1 Spacing

Spacing Conventions	
Situation	**Convention**
indentation	use xemacs and hit the tab key
{ and }	go on lines by themselves
arguments in function definitions and prototypes	go on lines by themselves
lists (ie enumerated values)	go on lines by themselves. try to line items up vertically
blank lines	use whenever whitespace is necessary

5.2.2 Comments

All code (including C, Fortran, Perl, Makefile) should have extensive comments.

General Comment Form	
inline	
c	`/* comment here */`
Fortran	`! comment here`
Make and Perl	`# comment here`
comment block	

c	`/* ------------ */` [4]
	`/* comment here */`
	`/* ------------ */`

Fortran	`!<spacing>------------`
	`!<spacing>comment here`
	`!<spacing>------------`

Make and Perl	`# ------------`
	`# comment here`
	`# ------------`

Comments may be placed anywhere to improve clarity...but there are several instances where comments are required.

[4] '-' may be replaced with '.' or '=' for lesser/greater emphasis respectively

See the appendix (A) for an example of the following.

Required Comment Areas	
situation	**required comment**
Programming Language	
Beginning of source file	Comment headers need to contain information like name, purpose, argument descriptions, return values, discussion, examples, ... This is to give the reader a general feel for the purpose of the routine.
Local variable declarations	The reason why variable declarations are placed on lines by themselves is so inline comments can fit to the right of the variable. Try to keep the comment length to just that line.
Executable Statements	Signals the start of the code.
Beginning of code blocks	Code blocks must start with a comment block describing what the code will do.
End of code blocks	Code blocks must also end with a comment block describing what has just been accomplished. This is done so that when editing, the developer does not have to scroll to the beginning of the code block to see what was done.
End of source file	For completeness.
Makefile	
Beginning of source file	Write a brief description of the purpose of the Makefile. Also, include one-line descriptions of the targets.
Variable declarations	Write a brief inline comment to the right of any variables
Before target/dependency line	Write a brief description of the purpose of the target/dependency

5.2.3 Common Functionality

All routines need to have common functionality in addition to the purpose of the routine.

Common Functionality	
situation	**description**
src (C, Fortran ...) code	
Error checks	All functions return an error code...and this error code is checked. If an error occurs, a message is printed and an error code is again passed up the calling chain. In this way, errors are passed all the way up to the top of the calling sequence.
Initialization	All UPS packages contain init routines. User accessible routines are in the aa package. Other packages have their routines in the utils directory. C interface routines check for package initialization.
Argument Values	Where possible, test input arguments for validity. For example, always have `default` statements in `switch` statements for bad input values.
Makefile see Appendix A.1	
Shell Scripting	The default shell is Bourne Shell
Standard Targets	A Makefile's first target is 'all'. This target is used for building whatever is required in that directory. The target 'clean' will remove any temporary files.
Recursion	Targets requiring targets from subdirectories will execute Makefile's in those subdirectories.
Error Checks	Commands are checked for success and error messages are printed for failures.
Non-Verbose	Commands are kept silent (but with appropriate echo's) so that errors (such as compilation errors) are easily visible.

5.3 Memory Allocation

Use `UPS_UT_Sm_malloc` when allocating shared memory and `UPSP_UT_MEM_MALLOC` (or the other flavor of alloc macros defined in `upsp_ut.h`) when allocating normal memory.

See the reference pages for `UPS_UT_Mem_get_item` for a discussion of issues such as `Performance Penalty`.

5.4 Error Checking

Use the `UPSP_ER_ASSERT` macro defined in `upsp_er.h` when testing for error conditions.

5.5 Environment Variables

Environment variables should be read in the ut package - upsp_ut_init(). Values should then be stored into the upsp_ut private struct. Then, other packages may use this struct to get values.

The exception to this is upsp_er_init() - which gets called before upsp_ut_init().

For now, mention pertinent environment variables in the "Variables" section of the comment header of the function. This might become cumbersome in that users will want a single page where all environment variables are listed. So, it might have to be changed in the future.

6 Porting

All good things must come to an end...eventually we'll have to port UPS to a new machine. This chapter contains information on what was needed to port UPS to different architectures. Having a history of the problems encountered in the past will show generic areas that probably have to be addressed during the current port.

6.1 Necessary Modifications

Some information cannot be determined prior to actual testing. Where possible, code should be designed to "break" when manual modifications must be made. For example, C source code might contain the following ifdef'd code:

```
#if defined (_SGI)
constant = 8;
#elif defined (_TFLOP)
constant = 7;
#else
compiler error.  you need to fine the correct types for this machine
#endif
```

So, when compiling on a new machine, the compiler will see a line it recognizes as invalid and produce an error.

The following files must be modified when doing a port to any machine:

- configure.dat

  Machine specific variable values must be edited. Go down through the file and make sure all the variables definitions make sense. New definitions for the architecture might need to be added.

- configure.pl

  The architecture option is predefined from this script. A "uname" is done and the variable "option" is set. Use the code for previous architectures as an example.

- doc/DeveloperGuide/ups_dg_porting.tex

  Edit this file and add the new subsection dealing with this port.

- src/cm/utils/upsp_cm_init.c

  Default optimization parameters are set to work for most cases. Architecture dependant changes can be set here.

- Compiling with gcc

  On many machines, make sure you have the correct modules loaded so that the correct gcc (and all other path-like variables) are being pointed to.

  One might have to set MPI_ROOT to point to the correct area.

- MPI cleaning

  Some system V machines tend to leave garbage MPI files if MPI does not exit properly. The location/existence of the following executables will vary:

  – mpiclean: cleans temp files

  – cleanipcs: cleans temp files

  - ipcrm: another option to clean temp files
  - ipcs: lists temp files

  /net/rogun/vol/proj/psp/installed/mahi/default/sbin/cleanipcs (comes with mpich/sbin location...

- long and long long: Dealing with i8

  We need to deal with i8 from our Fortran users. When compiling 64 bit, an i8 is the same as a C long. This is standard-compliant and thus easy to deal with. When compiling 32 bit, things get a little sticky...

  So, we are forced to use the following non-standardized items:

  - C datatype long long
    We expect a C long long to be the same as an i8. The compile will fail if "long long" does not exist. We also test for the size to match that of an i8.
  - MPI datatype MPI_LONG_LONG_INT
    This is in the MPI-2 standard...but not in MPI-1. Compiles that need it that do not have will fail.

  The following files deal with this issue:

  - configure.dat
    Set CM_DEFINES to -D_USE_PROTOCOL_LONG_LONG if the arch has MPI_LONG_LONG_INT and it is needed (32 bit addressing).
  - include/master_ups.h
    UPS_DT_Datatype_enum
    * 32 bit addressing
      long == long long == i8
    * 64 bit addressing
      long != (long long == i8)
  - src/dt/utils/upsp_dt_init.c
    Assign the size of "long long" and test for expected sizes.
  - src/ut/ups_ut_convert.c
    Convert to/from MPI_LONG_LONG_INT if _USE_PROTOCOL_LONG_LONG is defined (configure.dat).
  - testing/dt/fortran/test_sizeof.F
    Test sizes of ups datatypes corresponding to r4,r8,i4,i8 to make sure they are correct sizes.

6.2 Porting to SGI

We started on this machine[5] ...so perhaps we should call this section "Developing on the SGI".

Beyond that, we've ported UPS to an Irix64-bsaed dual-processor workstation. We note that workstation configurations within the context of a given vendor vary; therefore our experiences here may simply serve as a starting point. That said, the module system used by SGI greatly simplifies the work, and thus we simply list the differences in the "problems" section below.

- Help

  consult@lanl.gov 505-665-4444
  http://www.lanl.gov/asci/bluemtn

[5] Specifically, the ASCI cluster of SMPs running under the Irix64 operating system

6.2.1 Running on Nirvana and Blue Mountain

1. Getting on the Machine

   Go to the above website and get an account on the open theta.lanl.gov
   or nirvana.acl.lanl.gov/serenity.acl.lanl.gov machines. Use ssh to get onto any of the above front ends.

   From there, you'll need to use LSF (Load Fharing Facility) to get onto an interactive queue (for compiling and running). You cannot compile or run in the front ends. Type in

   ```
   theta:            bsub -n 4 -q threadq -Is tcsh -l
   serenity/nirvana: bsub -n 4            -Is tcsh -l
   ```

   On theta, the threadq does not limit the number of processes you can have (our Makefile system creates many sleeping processes). The other front ends do not have this problem. In any case, this gets you 4 processes.

2. Setting up the Environment

   The SGI uses "modules" to define your environment (eg path). I have the following in my .cshrc:

   ```
   if ( -f /opt/modules/modules/init/csh ) then
      source /opt/modules/modules/init/csh
   endif
   if (-f /opt/modules/modules/modulefiles/modules) then
      module load modules
   endif
   if (-f /opt/modulefiles/MIPSpro_default) then
      module load MIPSpro_default
   endif
   if (-f /opt/modulefiles/mpt_default) then
      module load mpt_default
   endif
   ```

   This sets which compiler (just use cc/f90) and MPI you will be using. In fact, you have to do have this on in your .cshrc in order to ensure you have the correct environment when mpi executables get spawned to other machines.

3. Runing with MPI

   After you have bsub'd onto a compute node, you can just type in

   ```
   mpirun ./<exectuable>
   ```

   This will run the executable on 4 processes.

4. Debugging

   To debug an MPI code, type in:

   ```
   totalview mpirun -a ./<exectuable>
   ```

   Use the middle mouse button to get options.

6.2.2 Running on SGI workstations

6.2.3 Problems on SGI

- theta front end and threadq

  As mentioned above, you have to use the threadq when compiling on theta. Our Makefile system creates many sleeping processes and overruns the process limit set for the normal interactive queue.

- MPI buffer limits

  We needed to do special things (message chunking, flow control) in order to get MPI not to crash when stressing the system.

- default vs. i8/r8

  When compiling using non-default integer sizes, you have to explicitly type constants so that they can be sent into UPS calls. Otherwise, f90 complains about it's inability to find a match when you call the UPS fortran interface.

- Optimized Compiler

  We cannot use the optimal compiler settings because they are incompatible from compiler version to compiler version. Must use just -O3.

- Number of processors

  In order to check some particular functionality, UPS tests create four parallel processes. If the computer has fewer than four processors, multithreading will be employed. This has the drawback of imposing some artificial synchronization which may mask runtime problems. Therefore, further testing using the number of available processes (less than four) should be done.

- mpt_default vs mpt_latest with multiple communicators

  It turns out that mpt_default has a bug that causes comm split/dup calls to fail when running on more than 4 boxs. As UPS creates comms, ups will fail on more than 4 boxs unless using mpt_latest.

  You'll have to set MPI_GROUP_MAX to something higer than its default of 32 (say 128).

6.3 Porting to TFLOP

This was our first port...

- Help

  pfay@lanl.gov (helpful expert on the RED machines)
  http://www.sandia.gov/ASCI/Red/usage/

6.3.1 Running on TFLOP

1. Getting on the Machine

   Go to the above website and get an account.

   To compile, you need to get to sasn100.

   ```
   ssh sasn100.sandia.gov -l <username>@lanl.gov
   ```

To run, you need to get to janus.

```
ssh janus.sandia.gov -l <username>@lanl.gov
```

2. Setting up the Environment

   I copied Pat Fay's (pfay) .cshrc (at least the part that defines path type stuff) and that seems to work for me.

   You'll use cicc/cif90 to compile your codes.

3. Runing with MPI

   To run, you have to be on janus (see above).

   ```
   yod -sz 4 ./<exectuable>
   ```

   This will run the executable on 4 processes.

4. Debugging

   To debug an MPI code, type in:

   ```
   debug -sz 4 ./<exectuable>
   ```

   This is a text debugger...kinda yucky till you get used to it (at least, so I've been told...I'm not used to it yet). I believe the web page has some pointers about it.

6.3.2 Problems on TFLOP

- MPI Communicator types

  On other implementations of MPI, the communicator is just an integer handle. On the tflop, it is a pointer to a struct. Fortunately, communicators gotten from fortran and from C point to the same thing which make our routines work correctly.

- MPI_Scatterv

  Every pe needs to know how many elements each pe is getting (in blatant disregard to the MPI1.1+ standard). So, I need to bcast that info out to every pe first.

- usleep

  Doesn't have it...so had to write our own using sleep. So, our minimum sleep time is 1 second. This is ok for us since this goes in our barrier_idle call which sleeps for long periods of time.

- mpif90.h

  The file in /usr/lib (or wherever the standard is) is written in fixed form. You can write one that satifies both fixed and free form easily...however, intel won't do it. Why? dunno. Fortunately, Pat Fay has one so we just include his.

  Also, the mpi parameters are in common blocks as opposed to being parameterized. So, needed to use equivalence instead of just setting parameters in testing/??/fortran/test_params.F.

  In a recent(Oct 20, 2001) development, looks like Intel has "fixed" mpif.h. "fixed" because although it does work with fixed/free form, it does not work with -i8. They need to specify KIND=kind(1_4) integer sizes instead of default. So, currently linking to one in my home directory...ugh.

- Fortran mod files

  This machine needs all the .mod files when compiling. Made a change so that every .mod file is moved over to the include directory.

- default vs. i8/r8

  Again, had to be careful when dealing with non-default integer sizes (like on the SGI).

6.4 Porting to Linux:naxos

Although we can now say "we have ported to linux"...that means less than it sounds like. We should really say "we have ported to one particular linux environment". When we port to a major unix architecture, we can feel pretty safe in assuming that certain tools will be consistently used on different machines of the same architecture. For example, on the SGI most people will be using the vendor supplied compilers and mpi. However, users on different linux boxes will probably be using differnent tools - which means different flags and behavior.

This will talk about porting to the linux box "naxos.lanl.gov" using the following tools:

- gcc
- Fujitsu F90
- mpich

The settings in configure.dat will probably have to change if we port to another linux box. We'll deal with that when we have to.

- Help

  As there is no one linux box, there is no single source for help. Try and find the person(s) who are using the tools and get them to tell you what they are doing. I talked to a couple of people (one for system stuff - Michelle Murillo:msqrd@lanl.gov and one for running mpi jobs - Tom Evans:tme@lanl.gov).

6.4.1 Running on Linux:naxos

1. Getting on the Machine

   Talk to Michelle and get an account. Then use your crypto-card to ssh over to naxos.

2. Setting up the Environment

   The default shell is bash2...deal with it. It is similar to tcsh...the big difference is how you define aliases/environment variables. I have the following .bashrc (which you have to manually source when you first get on the machine):

   The default location UPS looks for mpich stuff is correct on naxos. You might need to "setenv MPI_ROOT" to some other location if mpich is located somewhere else.

```
# .bashrc
# User specific aliases and functions

# Source global definitions
if [ -f /etc/bashrc ]; then
        . /etc/bashrc
fi
```

```
export LD_LIBRARY_PATH=${LD_LIBRARY_PATH}:/usr/local/FFC/lib
export PATH=/usr/local/tv4.1/totalview.4.1.0-3/linux-x86/bin:${PATH}
export PATH=/usr/local/bin:.:${PATH}
export XT_ARGS="-C -fn fixed -cu -j -rw -aw -s -sb -si -sk -sl 20000"

alias ls="ls --color"
alias la="ls -la --color"
alias rm="rm -i"
alias new="xterm $XT_ARGS -geometry 100x36 -e ssh-agent $SHELL"
```

`$(dollar not in .bashrc - needed for xemacs editing of this file)`

The first time you get onto naxos, you need to do some ssh stuff in order to get mpich to work via ssh.

(a) ssh-keygen
Enter your password

(b) cp .ssh/identity.pub .ssh/authorized_keys

After you ssh onto naxos (and "source .bashrc") you should use the alias "new" to get xterms. On the first xterm gotten by "new", you have to type in "ssh-add" and enter your password.

Now, things are ready to go (simple, eh?).

3. Runing with MPI

   With the above taken care of, this is pretty simple.

   Just `mpirun -np <number of pes> <executable>`.

4. Debugging

   Hahahah <tears streaming from eyes>...that is a good one! mpirun actually has some built in options for running under debuggers (type in "mpirun -h" for how to do this). The problem is that they don't work very well (at all?) when running fortran code. As all of our testers are in fortran...this leaves you with print statements.

   Serial (no mpirun) code can be debugged with totalview...for what that is worth.

6.4.2 Problems on Linux:naxos

- Lack of a Parallel Fortran Debugger

  With a good debugger, the port would have taken an afternoon instead of days.

- MPICH vs Vendor MPI

  MPICH strictly followed the standard for MPI_Init. You cannot pass null args to MPI_Init. So, our hack was to call the fortran-77 interface MPI_INIT if the user doesn't pass the calling args to UPS_AA_INIT.

  Also, other architectures have a better MPI_Abort where it actually does try and kill the other mpi processes.

- echo vs echo -e

  In order to simulate the behavior of echo on other architectures, we have to use "echo -e" instead of just "echo" on linux.

- Fujitsu Fortran Compiler

  When passing the same argument multiple times to a function, this compiler made copies before sending them to our Fortran interface. This caused problems when we were testing for "same argument" by testing if the memory addresses were the same. So, had to fool the compiler by using the Fortran-77 interface and sending in the first element instead of the whole array.

- gcc

  NULL is defined as a pointer to the address 0. So, the comiler complained when we were using it as an integer. So, had to change those places from NULL to 0.

- major bugs found

  The port did find some good bugs. Many places where the fortran intent was incorrect in the C comment header (which is used to generate the fortran interface). A lot of unused variables were found. Removed buggy special 1pe cases for several routines.

- POSIX

  On one flavor of gcc, _POSIX_SHARED_MEMORY_OBJECTS was defined yet the posix shared memory calls failed. This is a blatant disregard of the standard. Needed to add code that says "Use shared memory if the p_group is not UPS_CM_P_GROUP_SELF". Then we became more careful how the p_group gets defined in UPS_CM_Sm_get_item. (emu.lanl.gov)

- rand() vs RANDOM_NUMBER

  Got a seg fault when using the f77 rand/irand. Now, use the f90 RANDOM_NUMBER.

- argument not used still must not be null

  I had a test where a process was not reading anything (number of items to read is set to 0). So, it didn't allocate any space and sent that to the fortran call UPSF_IO_Dataset_read.

  Although this worked on other archs, it caused a core dump in linux. Seems that at least some space must be allocated. So, I init the buffer to -1's and make sure the values remain unchanged upon return.

  Although this still tests that no values in the buffer change, I am still disturbed that one must pass in a "allocated" address to the fortran routines.

6.5 Porting to Linux:lambda

lambda is a linux cluster on the open. It has been worked to look similar to the SGI clusters here...which is nice. So, it has modules, compilers, and totalview! Thus life is a lot easier.

- Help

  So, as this is a production cluster, you can email consult@lanl.gov.

6.5.1 Running on Linux:lambda

1. Getting on the Machine

   "ssh lambda" then "llogin -n 4" to get onto compute nodes with 4 processes.

2. Setting up the Environment

   I have the following modules set:

   - modules

- gcc_3.0.4
- LaheyFortran95Pro_6.1
- mpich_1.2.3.absoft-7.5
- totalview.5.0.0_4

The tricky part is making sure your mpich sets fortran symbols correctly (mpi_init_ as opposed to MPI_INIT/__mpi_init/...).

3. HDF GB size file test

   When running the parallel hdf tests on lambda, the "GB size file test" fails. I've contacted HDF and they are looking into it. Unfortunately, this bug causes a hang when running more than 1 pe. So, HDF tests run in effective serial mode. UPS tests are in parallel and test how users run the code...so that should be ok...

4. Runing with MPI

   One needs to use mpijob to run code on lambda. The format is just like mpirun, but you put mpijob in front. This is integrated into LSF similarily to theta:

   - `mpijob mpirun exec`
     runs exec on the number of processors you got (4 if `llogin -n 4`)
   - `mpijob mpirun -np 2 exec`
     runs exec on 2 processes

   There are probably other flags (like ptile) you can set.

   To run in batch mode, I get onto a compute node and then bsub from there: `bsub -n 16 mpijob mpirun <executable>`.

5. Debugging

   As there is totalview on this machine, just add the -tv flag to mpijob+mpirun command.

   `mpijob mpirun [args to mpirun] -tv [program] [args to program]`

6.5.2 Problems on Linux:lambda

- Clean up

  To clean up ipcs stuff, run `UPS/script/ipcclean.pl [username]`. This cleans up only the box you are on - but does not kill runaway execs.

  I wrote a script that wraps ipcclean.pl and killall called `UPS/script/cleanmachines.sh`. This ipccleans and killalls the most likely execs.

- linux:naxos

  A lot of the same probles on naxos apply on lambda. The port to naxos first made this simple.

- IO file syncing system

  Two things need to be sync'd so all processes have the correct state: mkdir and the file write. I solve this in the file open-read by doing a ls on the directory then doing a tail of the file. On the open_write, I just make sure that every process does the mkdir command.

- common scratch

  /scratch is common to 2 pe's. You can do /n/l4/scratch stuff, or use /netscratch/username (which is visable on all boxes)

- P4_GLOBMEMSIZE - not enough memory

  I set the following in my .cshrc

  ```
  setenv P4_GLOBMEMSIZE 20000000
  ```

  The default value is too small...so I just set it to something huge. This is probably bad if running on a bunch of processes so it will have to be tuned if that happens.

- Using FC_intel mpich needs iargc_/getarg_

  I needed to add -lPEPCF90 to the libs to create fortran execs.

6.6 Porting to Linux:intel1/bengal

These are new/experimental 64 bit linux clusters. They are not production machines...and you can tell. No modules, no mpi, old compilers, ...

- Help

  Yeah....right... Apparently, the sys admins are spoole@lanl.gov and parks@lanl.gov. I do not know if these are the correct people to contact. I get the feeling these machines are not really supported.

6.6.1 Running on Linux:intel1/bengal

1. Getting on the Machine

   You must go through a machine named ccn5temp4 to get to these.

   Once you can get on once, you can do some funky ssh stuff so you do not have to type in passwords...

   ```
   localhost % ssh-keygen -t rsa
   localhost % cd .ssh
   localhost % echo "ForwardAgent yes" >> config
   localhost % scp id_rsa.pub ccn5temp4:~/.ssh/authorized_keys
   localhost % scp config ccn5temp4:~/.ssh
   localhost % ssh ccn5temp4
   ccn5temp4 % ssh-keygen -t rsa
   ccn5temp4 % scp id_rsa.pub bengal:~/.ssh/authorized_keys2
   ccn5temp4 % scp id_rsa.pub intel1:~/.ssh/authorized_keys
   ```

   Permissions on .ssh files/directories must be for owner only (700/600).

   After this, you should be able to do the following from localhost:

   ```
   localhost % ssh -X -t ccn5temp4 ssh -X bengal
   localhost % ssh -X -t ccn5temp4 ssh -X intel1
   ```

2. Setting up the Environment

   Well, there are no modules....some stuff is in the typical place (/usr/bin). The fortran compiler on intel is in /usr/pgi/linux86-64/1.1/bin. Apparently, there might be other packages scattered around at the "/" level (eg /opt/intel/... for mpich).

3. Runing with MPI

   I have not done this yet. Apparently there is the problem with mpich being broken with its fortran symbols having two underscores. They need to rebuild mpich with one underscore.

4. Debugging

   I know of no debugger on these systems...printf.

6.6.2 Problems on Linux:intel1/bengal

- Limited UPS build

  I have only built serial no-fortran UPS. This seems to work...

- Probably the same stuff as other linux machines...

- Using FC_intel mpich needs iargc_/getarg_

  I needed to add -lPEPCF90 to the libs to create fortran execs.

- 64 bit compilation and LD_PRELOAD

  Although not available currently on lambda, it is on some experimental clusters (eg intel1). LD_PRELOAD is an environment variable that specifies libraries to be preloaded. It works fine for 32 bit compilation. For 64 bit compilation, it will fail (some error about not being able to find the 64 bit lib).

  When I try and manually load in the library (include it in the compile line), I get a segv....So, this does not work for 64 bit compilations.

- pgf90

  The version I am using is ancient (1.1-1) so the following might not apply... I cannot use the -Mchkfpstk compiler warning flag because I get an error about missing symbol __chk_stk_top.

  When I try and run Fortran tests, I get a segv. C-only tests pass...

- pgcc

  The version I am using is ancient (1.1-1) so the following might not apply... I get an internal compiler error when compiling with this. Objects are created and I can link, but programs segv...

6.7 Porting to Compaq Q - Alpha/HP/OSF

The Q system is currently in a state of flux (things are fubar if you ask me). So, the hurdles needed to be passed might disappear in the future - I hope so.

- Help

  consult@lanl.gov 505-665-4444

6.7.1 Running on Compaq Q

They (whoever "they" is) is trying to get the Q system to be just like SGI ASCI Blue (mentioned above). The process is getting to be the same...but at this time is a poor mix of Compaq and SGI standards.

1. Getting on the Machine

   Like the SGI, you get onto a front end (qalphafe, qidfe, c00, or some other front end) and the "bsub" onto the queue you want.

   Right now, "bsub" is "idlogin". Make sure you get enough processes (4) to run the tests. As of now on the qalphafe machine, you have to:

   ```
   idlogin -N 1 n 4 large
   ```

   Now, this shuts everyone else out of the large partition (2/3 of the machine). So, I would compile first with:

   ```
   idlogin -n 1
   ```

   and then try and run the tests after doing the larger idlogin command.

   For other alpha machines, you can try llogin and bsub and see what happens.

2. Setting up the Environment

   They are setting up modules like on the SGI.

3. Runing with MPI

   Currently, you need to type in:

   ```
   prun -n 4 -N 1 ./<executable>
   ```

   Other alphas might use dmpirun....

   Now, you have requested 4 processes (-n 4) on 1 box (-N 1). If you haven't gotten those resources from your idlogin command, this command will fail (something about insufficient resources).

   I needed to change the run command in configure.dat to run on a smaller number of processes in order to get things to run.

4. Debugging

   Totalview is there like on the SGI. Just replace "mpirun" with "prun" on the totalview line.

6.7.2 Problems on Compaq Q

- State of Flux

  They are still trying to get the Q machines to have the look/feel of the SGI. Try the SGI commands to do things and if that fails, flounder some more and ask the consultant.

- Division by O and timers

  Some MPI calls were so fast that the timers were returning a time of 0. So, if you divide by the time to do an operation, you get a division by 0 error. I needed to change the code to check for 0 time before doing a division.

- Compiler options on cc and f90 differ substantially

  configure.dat made the assumption that compiler options for the different languages will essentially be the same. This is not the case for the Compaq. I needed to modify configure.dat to have the ability for different options depending on language. This is probably a good thing since we might have a machine in the future that we wish to compile under different compilers (eg gcc and the vendor supplied compiler).

- gcc:MPI_ROOT

  Often, native compilers will know where to pick up mpi lib/inc. When using gcc, one can set MPI_ROOT to point to the correct place.

- gcc:stat

  Currently (2002/06/19), the gcc3.2.1 fails on linking source that had stat() calls - but gcc3.0.4 does not. Hopefully, the a patched version of gcc gets out some time. This problem does not appear on linux/sgi.

- Libraries

  Needed to include -lrt (the POSIX run time library) in the compile line.

- libuserd-HDF.so

  For some reason, when creating libuserd-HDF.so, you still need this library even if fortran was not built with it. I dunno why. If you do not have it, EnSight pukes with strange unsatisfied externs. So, there should only be the "!no_fortran" below. configure.dat:

  LIBS_F_WITH_C(no_fortran alpha_new — !no_fortran alpha_new) = -lfor

- Striped File System

  When putting files onto the striped file system (/scratch*), I have had problems with them being read in correctly. Specifically, when creating a .so. To fix this, it seems that if you add x permission to the file, it works. My guess is that the system treats executables differently. So, "chmod ugo+x" the .so after creating it.

6.8 Porting to Sun

Well, finally got to do this. Took a day. This port was different in that there was no official production machine to work with. I just did it on my local sun (and a couple of other suns).

- Help

  Well, it sure does help to have compilers installed...Other than that, there is no help :). /usr/lanl/gnu was very helpful in this regard.

6.8.1 Running on Sun

1. Getting on the Machine

   As there is no "production" sun, find one somewhere and ssh to it.

2. Setting up the Environment

   Make sure your path points to things that work. I was pointing to an "ar" that was broken. This resulted in a failure in the build that took a bit to track down.

3. Runing with MPI

   I used mpich that was already built. It was 32 bit...so I set 32 as the default build for suns.

4. Debugging

   Surprisingly, I didn't have to use a debugger. Things worked. This is nice because as a test, I tried to bring up a gui debugger ("debugger" which pointed me to "workshop -D" which crashed) and dbx (which just core dumped right away...saving me some time).

6.8.2 Problems on Sun

Besides the above, I hit the following.

- Bit addressing

  default compilation is 32 bit addressing. To get 64, you need to set add "-xtarget=ultra -xarch=v9". I decided to have the default to be 32 bit because many tools (EnSight, mpich) are built 32 bit. To get 64, just configure with "-32 +64".

- Shared memory

  The place to "shm_open" virtual shared memory files on the sun is "/". This is different than other architectures where the underlying posix shared memory functionality actually creates files and thus the directory must be writable. So, on sun, the default location is "/" and on other machines, the default location is "/tmp". This is modifiable by setting the environment variable UPS_TMPDIR_SM (as explained in the comment header/reference manual for UPS_CM_Sm_malloc).

- Atomic Functions

  UPS_CM_Sendrecv (when shared memory optimizations are turned on) uses system functions for atomic operations (atomic_add_32). While these are prototyped in /usr/include/sys/atomic.h, Sunlung and I could not find the library containing the funcs anywhere. So, we just don't use sendrecv optimizations and default to using MPI_Sendrecv.

- Other

  Typical problems that will occur with any ports. Many of these are designed via #ifdefs to automatically pop up when moving to a new architecture. Things like

```
#if defined( _IRIX)
foo = 1;
#elif defined( _SUN )
foo = 2;
#else
sentence that will generate a compiler error
#endif
```

  So, when you move to a new architecture, you cannot compile until you set things correctly. The fixes included: compiler warn flag, optimization flags, module include path, addition libs that must be linked with, size of long/long-long/pointers.

6.9 Porting to AIX

I worked with Susan Post with this one...so I didn't have to surmount the learning curve I have had to do on most machines. The "AIX" I ported to is frost.llnl.gov.

- Help

  Well, Susan and llnl email help (lc-hotline@llnl.gov [925-422-4531]).

6.9.1 Running on AIX

1. Getting on the Machine

   There is an ssh version conflict between theta.lanl.gov and frost.llnl.gov (secure machines bluemountain to white are fine). So, the following examples tell ssh/scp to use protocol 1.

   When the version conflict is over, these options should be removed.

   (a) get onto theta and get a kerberos ticket

   (b) `ssh -1 frost.llnl.gov`

   To copy files, you would type:

   ```
   scp -oProtocol=1 <file> frost.llnl.gov:<path>
   ```

2. Setting up the Environment

   There are no modules on frost, so location of stuff is in the standard places (eg /usr/loca/bin for compilers).

3. Runing with MPI

   You must set `MP_RMPOOL` to be 0 (interactive) or 1 (batch). If you set to batch, you have to use psub to submit jobs. I have not learned how to do this yet. I have been able to run on the interactive machines (blue, frost, ice).

   Then you can either:

   ```
   poe -nodes 1 -procs 4 <exec>
   mpirun -np 4 <exec>
   ```

   I use the "poe" methos since it works with the debugger (see below).

4. Debugging

   You can use totalview (although displaying is slow since you must display through ssh).

   I could not get totalview to work with mpirun, but could with poe:

   ```
   totalview poe -a -nodes 1 -procs 4 <exec>
   ```

   If you run with more than 1 process, poe prompts you for the name of the exec and any arguments.

   When compiling with Fortran and using the C preprocessor, the compiler generates temporary files. In order for totalview to display source, you need to keep these temp files using a -k option when compiling. The name appears to be:

   ```
   F<old file name>.f
   ```

6.9.2 Problems on AIX

- Configure.dat

  Things like name of C compiler, -i8-r8 options, blah blah...not too bad.

- Defines in Fortran

  The define flag for fortran is "-WF,-P,FOO -WF,-P,BAR ...". Noooo, nothing so simple as "-DFOO -DBAR". That would be too easy. Thanks to gmake, the pattern substitution was easy to do in configure.dat.

- _F2C_LOWER_CASE

  Objects with mpxlf are lower cased without an underscore. Needed to add that option to configure.dat and prototypes in .h files (include/master_ups.h, src/cm, experiments/pt2pt_scheduling, test/aa—cm—dt—gs—ut).

- abs() vs (d—i)abs and Fortran

  Replaced the last few dabs/iabs with abs. Let Fortran figure out which one to use...

- Bit Addressing Flags

  Apart from setting bit addressing for the compiler (-q32 or -q64), you must set it for ar as well (-X 32 or -X 64). For ar and for the compilers, and perhaps other things, you can specify OBJECT_MODE to be 32 or 64 as well. This will be overridden by any flags given.

- Fortran sleep system call

  I had been using `call sleep(int_1)` to sleep for 1 second. On aix, the call must be `call sleep_(int_1)`. If you use the first form, you compile and link. However, you hang in the sleep call. So, I made a UPS wrapper `UPS_IO_Sleep( double seconds )` that calls the appropriate sleep functions.

- 64 bit integer constants

  When compiling with 32 bit addressing, the compiler pukes on 64 bit constants (well, truncates them to 32 bit). So, I had to add the suffix llu:

  ```
  unsigned long long b = 0xd0ab0f0002c00600llu;
  ```

  If this does not work, I could do the following:

  ```
  unsigned long long b = (b = 0xd0ab0f00) << 32 | 0x02c00600;
  ```

- STDOUT output

  When building with the mp compiler suite (eg mpcc), all output to stdout is prepended with some process information. The tests that do a diff on output to test correctness will fail because of this (as the original output might have been run on a machine without this prefix).

  If you set MP_LABELIO to be no, you will not get this.

- compiler to use

  The main compilers for serial builds are xlc, xlC, xlf. For parallel builds, the names are mpcc, mpCC, mpxlf). The threadsafe versions have a _r suffix. I used these (in order to coexist with HDF).

- tools:libuserd-HDF

  For some reason, EnSight pukes at startup if I give it the location of the parallel lib (serial works fine). I'll still build it since my tests run and pass...

- mpirun vs. poe

  We are supposed to use poe to run mpi apps. I found you could use mpirun as well. I set environment variables (MP_PROCS and MP_NODES) instead of the poe flags because otherwise poe asks me interactively for that info.

- build directories

  I do not know if there is an official scratch space. So, I just built in my home directory. There is no simple absolute path that is consistent across machines (eg /home/lmdm). I have /g/g14/lmdm on white and /g/g20/lmdm on frost. This name inconsistency makes some automation annoying.

- deallocate

  I did not free some allocated Fortran arrays. When the routine was reentered, the system gave an error about not being able to allocate the space. When I deallocated the arrays, things worked.

- HDF

  HDF is located at /usr/local/hdf5/.... You have to find the flavor you want under there.

6.10 Porting to .....

- Help

6.10.1 Running on ...

1. Getting on the Machine
2. Setting up the Environment
3. Runing with MPI
4. Debugging

6.10.2 Problems on ...

-
-

7 The UPS CVS Repository

The UPS project utilizes the Concurrent Version System, CVS[2] for software management and version control. CVS allows several developers to contribute to the UPS software library in an organized and coherent way.

Each developer works within his own directory space, and then "commits" source code to the repository. This doesn't, however, prevent problems from occurring. Instead, a set of rules must be followed by each developer. These rules are not enforced by CVS.

The UPS CVS repository is used to maintain working, tested code. It should not be used as a backup system.

7.1 Version Number

The version number of UPS is found in several different places (and in slightly different forms [an example is given in square braces]):

- configure.dat variable [`PRODUCT_VERSION = v-02-07-03`]
- exec variable [`UPS_VERSION = 20703`]
- Compiled Function Name [`ups_version_20703_`]
- CVS Tags [v-02-07-03]
- Installation Directory [v-02-07-03]
- UserGuide/DeveloperGuide [v-02-07-03]

The value of `PRODUCT_VERSION` and `PRODUCT_VERSION_PREVIOUS` in configure.dat dictate all of these. Make sure it is correct.

7.2 Committing code to the repository and installing/releasing

Your work should be committed to the repository on a regular basis, where the time interval is defined by the amount of work you have done, the location of this work (several different directories adds to confusion), and the amount of work done by others.

The first step to committing source to the repository is to have your code or idea reviewed by a peer. People may commit code so that others may have access for the purpose of review. When using this method of peer review, the developer should use the `UPS/peer_review/<username>` directory. Upon committing, you will be asked to enter the name of the peer reviewer. You can just put your email address. When it has been reviewed, then just remember to remove it from the repository.

After this is done and your code is written, ensure that your version of the library compiles and successfully passes all of the tests, including those tests that may not appear related to your changes and additions. Recompile the library and run all the tests. Pay special attention to any warnings/remarks you get - do they make sense? Make sure that any changes you have done have not introduced any new bugs in other areas.

Following is the procedure for fixing a bug and committing to the repository. This includes steps that ensures new files you may have created are included, allows you to investigate potential code conflicts *before* committing, and sees which files have been altered by others and which will therefore be changed in your copy of the library. These steps are not enforced by CVS; instead it is the responsibility of UPS developers to ensure that they are followed.

1. **Assign Debugger**

(a) Discuss with team who should fix bug.

(b) Modify and commit `peer_review/<developer>/ToDo.<developer>` to reflect assigned work.

2. **Verify Bug**

   Check out version of UPS that breaks for user and run their test code.

3. **Debug**

   (a) Create test case (to be added to `testing` directory) that recreates bug. Successfully pass this test with debugged code.

   (b) Run full test suite for all packages while fixing bug. This insures no other bugs will be created.

4. **Commit Debugged Code**

   (a) Get a peer reviewer to O.K. your changes.

   The following steps can be followed to get a summary of the changes that have been made. Looking at your changes as a whole can help find potential problems (eg missed modifications, debug statements that should be removed, conflicts, ...).

   i. `cd <local location of UPS>/UPS`

   Gets to top directory. You want the cvs commands to work on your entire working directory (so nothing slips in the cracks)

   ii. `cvs -n update -d -P | sort`

   Gets a listing of the status of all the files in your working directory without actually doing any changes. Pay careful attention to:

   `?` : unknown file. Have you forgotten to add this file to the repository (via `cvs add`)?

   `C` : another user has modified a file I am working on. Are you safe to update your area? Do a `cvs log <file> | more` and talk to the author to make sure.

   `U` : another user has modified a file I am not working on. Will the changes this user has done affect your work? Or will you affect his? Do a `cvs log <file> | more` and talk to the author to make sure.

   iii. `cvs update -d -P | sort ; gmake clean; gmake`

   Get the most recent version from the repository. Recompile and rerun the tests to make sure all the testers still pass

   iv. `cvs diff -D now > diff` (on one window)

   Gets the complete list of changes you have made. Look over these changes for any possible errors. You use this when you do a commit on another window. Don't expect to remember all the changes you have made....use `cvs diff` as a reminder.

v. `cvs -n update -d -P > check`

This creates a template for your commit log message. The form is:

```
A <added    filename with relative path from checkout>
M <modified filename with relative path from checkout>
R <removed  filename with relative path from checkout>
<3 spaces><synopsis of changes>
<repeat as needed>
```

Figure 1: Example of CVS Commit Log

Bring up the file `check` in another window and modify it to create your commit log.

vi. Discuss changes with Peer Reviewer.
You might also want to commit files to your peer review directory so that the peer reviewer can look at them.

(b) Add discussion of major changes since last release to README.
If your commit is substantial, make a note of it in the README.

(c) `cvs commit -F check`

5. **Release Debugged Code to Users**

(a) `gmake release_notes`
Review all_changes.txt and see if anything needs to be added to the README (eg major changes since last release). If so, commit the README when finished. Use the "major changes" as an email to send out to ups-users@lanl.gov.
You will need to decide the level of the release at this point.
The form for release numbers is:

```
v-<Major Changes>-<Changes>-<Minor Changes>_<Subrelease>
```

- Major Changes
  Adding new package(s).
- Changes
  Adding extensive new functionality to package.
- Minor Changes
  Bug fixes or minor functionality changes.
- Subrelease
  Temporary changes for a user to test.

In the "gmake install_dev" stage, you will be asked to enter this name which will be used for the install directory name and for CVS tagging.

(b) `gmake full`
Builds all necessary files (libraries, includes, docs, ...) If cvs accessible, this command will remind you how to turn off commits before doing anything.
This cleans up your lib/include directories and builds all the necessary files for a basic install (as specified by OPTIONS_FULL in configure.dat).

(c) If you wish to include additional libraries, you can then:

i. `configure.pl go <other options>`
ii. `gmake clean [this will clean so the new lib can be built]`
iii. `gmake`

Repeat the above steps for any other additional libraries.

(d) `gmake install_dev`

Just answer the questions.

"gmake install" does the following functions:

- make sure source is up to date (if cvs accessible)
- copies files to installation area
- creates latest link
- installs web pages
- cvs tags repository (if cvs accessible)
- prints uninstall instructions

(e) Creating a source distribution

If you wish to create a source distribution tar file, you can execute the command "gmake dist".

8 Adding Parts

The following are examples that give an idea of what has to be done when new parts are added. For all cases, try and follow the example of other tests/files/packages for how to do something.

8.1 Adding a New Test

This might be done if you wish to stress some functionality (with the gs package as an example).

1. **Create necessary files**
   - `gs/fortran/test_new.F`: New Fortran test function.
2. **Modify necessary files**
   - `gs/fortran/prog.F`: Main fortran program calls tests.
   - `gs/fortran/Makefile`: Must compile new test (remember to add Fortran dependencies)

8.2 Adding a User Accessible Function

When a new function is written for the user, the following steps have to be done (with the gs package as an example).

1. **Create necessary files**
   - `UPS_GS_NEW.c`: Fortran77 Interface
   - `ups_gs_new.c`: C Interface (generates Fortran interface)
   - `internals/gsi_mpi/upsi_gs_new.c`: Internal Interface
   - **New test**: see **Adding a New Test** (section 8.1 page 73)
2. **Modify necessary files**
   - **Constants**
     `master_ups.h` contains most constants. You will have to add entries to the following enums:
     (a) `UPS_Error_enum`
     (b) `UPS_Code_location_enum`
   - **Makefiles**
     For the added files, you need to edit the Makefiles in order for them to be compiled. Remember to add dependencies of fortran files because of the fortran `use` statement.
   - **Mod files**
     Edit `UPSF_GS_MOD.F` and add UPSF_GEN_GS_NEW to the `use` list.
   - **Prototypes**
     All C functions are prototyped. Prototypes are kept in:
     – `master_ups.h`
     – `upsp_gs.h`
     – `upsi_gs.h`

8.3 Adding a New Package

As an example, the zz package is being created. For all files, look and see what other packages did and mimic them.

1. **Create necessary files**
   - `src/zz, testing/zz`: see **Adding a User Accessible Function** (section 8.2 page 73) Add all the src and testing files in a way similar to other packages.
   - `doc/UserGuide/ref/ups_ug_zz_refman.tex`: short blurb introducing the zz package's reference section.
2. **Modify necessary files**
   - `src/Makefile`: Add zz package to list.
   - `testing/Makefile`: Add zz package to list.
   - `configure.dat`: Variables need to be updated:
     - `DEFINES`: All defines
     - `INCLUDE_PATH`: Location of all .h files
     - `UPS_INTERNALS`: Name of all internals name
     - `LIBS`: All libs
     - `ZZ_DEFINES`: Package specific defines
     - `ZZ_INCLUDES`: Package specific includes
     - `ZZ_INTERNAL`: Package specific internal name
     - `ZZ_LIBS`: Package specific libs
   - `UserGuide`: UserGuide Information
     Look for places where other packages are mentioned.
     - `ups_ug_introduction.tex`
     - `ups_ug_packages.tex`
     - `ups_ug_reference.tex`
     - `header2latex.pl`
   - `Web_page`: The web pages
     Look for places where other packages are mentioned.

8.4 Adding New Aux Product

In general, when adding something, copy/follow-by-example files that already exist. A new aux product is no different:

1. `mkdir aux/<aux_dir>`
2. `cvs add <aux product>.tar.gz`
3. Copy then modify `Makefile` from another aux dir
4. Copy them modify `configure_local.dat` from another aux dir
5. Modify `configure.dat:AUX_DIRS` to point to new directory

8.5 Adding New Tool

In general, when adding something, copy/follow-by-example files that already exist. A new tool product is no different:

1. Copy/Modify existing tool files to new tool directory.
2. Modify `configure.dat:TOOLS_BUILD` variable

8.6 Adding Ability for Someone to Access UPS

"Access" to UPS means different things. This can range from just getting emails notifying when a new version has been installed all the way to full write permission into the UPS CVS repository.

The following are the different levels of access in order of immersion (and the steps needed to be performed):

1. Adding to the ups-users mailing list:

   This will inform the user when a new version has been installed. Access to UPS will be limited to what is contained in the /usr/projects/ups/<version> directory. This includes libraries, scripts, and documentation.

   See the "...modify the ups-team/ups-user mailing lists" section under the "How do I" chapter.

2. Adding to the ups unix group:

   This will allow the user to check out the ups product. They will be able to look at the most current source and build UPS. They will also have manual write access to the repository/installation directory. So, one must be careful of "rm -f *" commands...

   In theory, to add a user to the "ups" unix group, you should do the following:

   (a) telnet to register.lanl.gov
   (b) Edit a File Sharing Group Name (8)
   (c) enter "ups"
   (d) Set File Sharing Group Attributes (6)
   (e) Add Members (3)
   (f) enter the username(s) of the people to add
   (g) hit return a bunch of times to exit

   This should then propagate to all the systems (open and closed).

   This works for theta/bluemountain...but doesn't for nirvana. That is to say the above works for some systems and not for all. So, I do the above and then email the consultants (consult@lanl.gov) and tell them that you have added users via register and need to make sure they are added to theta, bluemountain, nirvana, qalpha, ... whatever.

3. Adding to the ups-team mailing list:

   This will inform the user about the details of ups progress. This is important if the user needs to know whenever a commit is done (as the user will now get commit log emails) or if the user wishes to hear the innermost secret discussions regarding UPS.

   See the "...modify the ups-team/ups-user mailing lists" section under the "How do I" chapter.

4. Adding to permission_cvs.pl

   This allows the developer (note the change from the word “user” to “developer”) full CVS write access. The prerequisite to this is that the developer should have read the DeveloperGuide and understands the rules for committing. All the above levels are included if given this level.

   Check out CVSROOT and edit permission_cvs.pl and add the developer’s name to the permission list. Remember to commit again.

   People will have different needs and it is important to give them the correct access.

9 How do I

This section is intended as a reference to answer the question, "How do I...". If you do not find what you are looking for here and you figure out how to do whatever it was you were trying to do, then feel free to add it to this section.

9.1 CVS

9.1.1 ...get rid of empty directories and get files that others have a dded

Whenever I do an update (unless I have a good reason not to), I want to prune any empty cvs-controlled directories and get files/directories that others have added to the repository.

To do this, you need to:

1. `cvs update -d -P | sort`

Note: if you have any temp files in a directory you want pruned, you have to go in by hand and remove them before cvs will prune the directory.

9.1.2 ...look at an older version of an existing file with cvs?

To get the latest version of a file that was committed to the repository, you can just:

1. `cvs update -p [file] > [file].orig`

This places the latest committed version of file in file.orig.

9.1.3 ...look at a specific version of a file

To get a specific version of a file, you can just:

1. `cvs update -p -r [version] [file] > [file].orig`

If you need some prompting on what version you actually need, you can just `cvs log [file] | more` to get a listing of the log messages for that file. Then you pick the version you want out of that file.

9.1.4 ...look at an old version of a file in a directory that doesn't exist anymore

Suppose you need to look at an old C tester and the C directory doesn't even exist anymore. First you have to get the directory back:

1. `cvs update -d [directory name]`

If you are missing directories `b/c/d` and you wish to look at file `b/c/d/hmm.c` you can do the following to get the directory:

1. `cvs update -d b`

Now, to more the last non-empty version (because the file was removed, the last version is empty), you have to find the last committed version. Do a log command on the file and get the pre-removal version. **before** it was removed.

1. `cvs log [file] | more`
2. `cvs update -p -r [version] [file] > [file].orig`

When you are done with the directories, you can "prune" them again by:

1. `cvs update -P [directory]`

The directory must be recursively empty (except for other subdirectories and, of course, the "CVS" directory(ies).

9.1.5 ...check out a previous installation

Suppose a user has a problem with an old version of UPS (and cannot, for some reason, get the newest version). If you want to get access to the source that created this version, do the following:

1. `[get version number that user has]`
2. `cd [temp space]`
3. `cvs co -r [user version converted into v-##-##-##] -P UPS`
   Example: `v-01-02-01`

Note: you will not be able to modify this source and save it to the repository. To do this, you need to make a cvs branch.

9.1.6 ...add a directory tree to the repository

Suppose you wish to add the directory tree a/b/c and all the files under it to the directory UPS/testing (which exists in the repository).

1. `cd a`
2. `cvs import UPS/testing/a vt rt`
3. `[enter log message explaining your import]`

If you had created this directory tree in your working repository, you need to remove it and then update to get it under cvs control. Suppose, you had created the tree in the place you had it installed.

1. `cd UPS/testing`
2. `rm -rf a`
3. `cvs update -d -P a`

9.1.7 ...remove a directory tree from the repository

Suppose you wish to remove the directory UPS/testing/a and all the files it contains from the repository.

1. `[remove by hand all the files (ignoring the CVS directories) in a and its subdirectories]`
2. `cd UPS/testing`
3. `cvs remove a`
4. `cvs commit`
5. `cvs update -d -P a` (prunes the empty directory a)

9.1.8 ...turn off commits

There will be times when you need to turn off commits (eg to test something or to do something with CVS that cannot be interrupted by another user's commits). You can do the following:

1. `cd $$CVSROOT/CVSROOT/`
2. `[edit permission_cvs.pl and have only your name as a valid user]`

   Due to permissions, you might have to copy permission_cvs.pl to a new file, remove permission_cvs.pl, and copy back to permission_cvs.pl.
3. `[do you testing]`
4. `[edit permission_cvs.pl and restore the valid users]`

9.1.9 ...look at the differences you have made in a file as a whole

If you just type in "`cvs diff <file>`", you can get the differences you have made to a file. However, it will be in simple diff format (without any context). If you wish to see the whole file (with "ifdef" control sequences), you can do the following:

1. `cvs diff --ifdef=new <file>`

   This sends the ifdef'd code to STDOUT.

9.1.10 ...check the differences between a file and the most recent checked in version

When you type in "cvs diff [file]" you only get the changed you have made to the file. Suppose someone had committed a version and you want to see the differences before you do an update. Do the following:

1. `cvs diff -D now [file]`

9.1.11 ...move the whole repository to a new location

There might be a need to move the whole repository from one directory to another directory. This is how it can be accomplished:

1. Have everyone commit (or copy their modified files to another area).

   Once you move the repo, they will not be able to access the old repo any more.
2. Change the permissions of the Repository so that only you have access.

   You want the repo to be unchanging.
3. `tar cf ups.tar Repository`

   Create a tar file for the move (and for backup purposes).
4. `cp ups.tar <new location>`
5. `cd <new location>`
6. `tar xfp ups.tar`

   This will create Repository in the new area. The "p" option will maintain all the old permissions.

7. move any other files from the old area to the new one.

   There might be other files (specifically user accessible directories) that you wish to move.

8. Set the correct permissions for the repository.

   (a) directories need to be group writable with the group sticky bit set.
   This is so when someone enters a directory and creates a file, it will have the correct group ownership.

   (b) files should be group writable, in general.
   Plain files under CVS control need only be group readable (but directory permission needs to be correct).

9. Everyone must change all references of the old area to the new one in their environment.

   Typical places are the **CVSROOT** environment variable and the user's **PATH**.

10. The mover needs to set hardwired paths to correct areas.

    At this point, the mover can checkout a fresh copy of the product and make commits.

    There were a few places in some text (latex/html) files that have the hard path in them. Also, one needs to edit `configure.dat` and change the `UPS_INSTALL_DIR` variable. This variable cannot use `CVSROOT` because some installations will be on machines without cvs control.

11. Everyone needs to delete their working version and checkout a fresh one.

    The best way to check out the product is to :

    ```
    cvs co -P <product>
    ```

    The "-P" will prune empty directories.

    After the mover has fixed a few things (previous step), everyone is ready to checkout a new version. For those that had modified files, they will need to copy them to the fresh checkout.

9.1.12 ...abort a commit

Sometimes you'll be doing a commit and realize that you do not want to continue.

The commit has already processed any files for which you have successfully entered a log message. To stop the commit of the files in your current log message and to the rest of the files to be processed for the commit, simply:

1. Exit the editing session ("`:q`" or "`:q!`" in vi)
2. Answer "`n`" to the continue question.

9.2 Perl

9.2.1 ...edit a whole bunch of files with a simple search/replace

I'll start off by saying the whole command and then explaining the parts:

1. `find . -name '*.[ch]' -print | /bin/xargs perl -pi -e 's/UPS/ups/g'`

find .	find recursively from the current directory
-name '*.[ch]'	any file that ends in ".c" or ".h"
-print	print the files to stdout
|	pipe the output of the previous command to
/bin/xargs	a program that takes each line of output and send it to
perl	an awesome tool
-p	take each line of input and echo it first
i	edit the file in place
-e	with the expression
-s/UPS/ups/	before printing the line, replace "UPS" with "ups"
g	as many times as you have to in a line

As another example, suppose you want to replace "UPS_CM_ERROR" with "UPS_ERROR_CM". The trick is, you do not wish to replace "UPS_CM_ERROR_A" with "UPS_ERROR_CM_A". That is, you wish to replace, but check word boundaries. You wish to make this change on all ".c" and ".h" files in src.

You would :

1. `cd src`

2. `find . -name '*.[ch]' -print |`
   `/bin/xargs perl -pi -e 's/(\b)UPS_CM_ERROR(\b)/$1UPS_ERROR_CM$2/g'`

()	remember the contents of the parenthesis
backslash b	word boundary
$1	the 1^{st} parenthesis's contents
$2	the 2^{nd} parenthesis's contents

9.3 Make

9.3.1 ...eliminate most errors I am having with make

Make sure you are using gnumake.

9.3.2 ...switch from creating a debug version to an optimized version of libups.a

In the top level directory (UPS), there is a perl script called configure.pl. It parses configure.dat to make make.inc (which all makefiles include). To temporarily remove "debug" from the default options, you can just type in "configure.pl go -debug" which stands for "minus debug". Remember that if configure.pl is run again without the additional option, the default values are used.

To permanently change the options (so that you can just type in "configure.pl go") you can edit the "CONFIGURE_OPTIONS" line inside configure.dat to pick whatever options you wish.

9.3.3 ...easily run the tests given that I have 2 libraries

You do not have to recompile the source again, but you will have to recompile the testers.

1. `cd UPS`
2. `[edit CONFIGURE_OPTIONS in configure.dat for your desired options]`
3. `configure.pl go`
4. `gmake bld_run_tests`
5. `[edit CONFIGURE_OPTIONS in configure.dat for your desired options]`
6. `configure.pl go`
7. `gmake bld_run_tests`

9.3.4 ...install ups, but just copy the files

There might be times when you wish to copy all the needed files to the installation area...but you do not wish to do the other things an install does (like cvs tagging and changing the latest link). This is useful when installing 1 set of libraries compiled under the default compiler and another set compiled under the latest compiler.

Follow these steps:

1. `[set your compiler to new compiler]`
2. `gmake full`

   If cvs accessible, this command will remind you how to turn off commits before doing anything. Docs and release notes will be created if they do not exist. If you wish to create new versions, you must remove the old versions by either removing the files or by doing a `gmake distclean`.
3. `gmake install_copy INSTALL_NAME=v-##-##-##<name>`

   For example, `INSTALL_NAME=v-02-01-03_MIPSpro7.3`
4. `[set your compiler to the default compiler]`
5. `gmake full`
6. `gmake install`

   Note: install is done second because of possible tagging (which affects UPS_VERSION).

9.4 SQA

9.4.1 ...get others to review my code I wish to commit

Suppose you have made changes to a file and are ready for it to be reviewed by peers. There are several possibilities for getting that code to the reviewers. You could email them the files, leave it group readable somewhere, or print it out and give them hard copies.

We have a directory called "peer_review" in the top level directory specifically set up for this purpose. You will place the files you wish reviewed under the subdirectory specified by your username.

The following is a method that puts your code into the repository so that reviewers can have access to it.

Suppose you have changed ups_gs_setup and the internals file upsi_gs_setup.c and wish them to be reviewed.

1. `cd UPS`
2. `cvs update -d peer_review/<username>`
   [this gets the subdirectory if it didn't exist]
3. `cp src/gs/ups_gs_setup.c peer_review/lmdm`
4. `cp src/gs/internals/gsi_mpi/upsi_gs_setup.c peer_review/lmdm`
5. `cvs add peer_review/lmdm`
   [this will add everything in that subdirectory to the repository]
6. `cvs commit peer_review/lmdm`
7. `[get the peer review done - when done, remove files]`
8. `rm peer_review/lmdm/*.c`
   [this removes the .c files]
9. `cvs remove peer_review/lmdm`
   [this cvs removes the files]
10. `cvs commit peer_review/lmdm`

9.4.2 ...use Insure (compile-time/run-time code checker)

Insure is located on the nirvana, theta, and bluemountain machines. You can use Insure to find many bugs in C/C++ code. It will also so some checking (e.g. memory problems) for other code (e.g. Fortran) but it isn't as thorough.

Currently, Insure4.1 seems to work and Insure5.1 does not. To run Insure4.1, you need to set the following (which I do in my .cshrc):

```
setenv PARASOFT /users/<USER_NAME>/insure
set path = ( $path ${PARASOFT}/bin )
setenv LD_LIBRARY_PATH ${LD_LIBRARY_PATH}:${PARASOFT}/lib
setenv MANPATH ${MANPATH}:${PARASOFT}/man
```

And then I make a link in my home directory of:

```
ln -s /usr/local/packages/insure++ ./insure
```

Set the following in your .cshrc to load the new insure5.1 module:

```
if (-f /opt/modulefiles/insure++5.1) then
   module load insure++5.1
endif
```

To build ups with insure, just add insure to CONFIGURE_OPTIONS in configure.dat. Due to a set number of user licenses, you cannot do a large parallel build. So, a smaller build is performed (which takes much longer).

Then you just run your executable. It has a fairly intuitive GUI. I would suggest printing out the insure documents (found in the same location as the libs/executables).

9.4.3 ...use Purify (run-time code checker)

You must be on a machine that has purify (and purify must be in your path). Purify instruments your executable so, unlike Insure, you can easily toggle purify on/off. Typically one might do the following:

1. gmake (to build library/run tests)
2. configure.pl go +purify (will run purify)
3. cd testing/cm/fortran (go into test you are interested in)
4. rm xtest_cm (remove test so that new test will run via purify)
5. gmake && gmake run (builds purify'd exec and runs it)
6. vi purify_log.txt (examine the purify output)

9.5 Xemacs

9.5.1 ...change file type mode

Xemacs tries to get the correct file type (so, for example, it knows how to indent when you hit the tab key. However, sometimes it doesn't do very well. For example, to change to modern Fortran mode, you can do the following:

1. `<esc>-x`

   This activates command line mode.

2. `f90-mode`

What you might wish to do is add some lines to your .emacs file so that certain files are always input in a certain mode.

```
(autoload 'f90-mode "f90")
(setq auto-mode-alist (cons '("\.F$" . f90-mode) auto-mode-alist))
```

9.6 Misc

9.6.1 ...recover files from a NetApps file system

Fortunately, the home area on Blue Mountain is a netapps. This makes file recovery simple. Say you deleted a file that you can't get back via cvs commands. Here is the process

1. `df -k [the directory you lost the file]`

   This tells you the actual filesystem you are mounting.

   If the output was "mother:/p0/local/irix/default", the main filesystem is "mother:/p0"

2. `cd /n/mother/p0/.snapshot`
3. `cd [last time period you need]`
4. `cd [path to missing file]`
5. `cp [file] [your local area where you need file restored]`

9.6.2 ...modify the ups-team/ups-user mailing lists

The ups-team@lanl.gov and ups-user@lanl.gov mailing lists are maintained by the LANL's list manager. To modify these lists, do the following:

1. Go to `http://listman.lanl.gov`
2. Click on modify.
3. Enter "ups-team" or "ups-users" for the list name.
4. Enter the password and click on go.

References

[1] R. Barrett and M. McKay Jr. UPS user's guide and reference manual. Technical Report LA-UR 99-2857, Los Alamos National Laboratory, 1999.

[2] Per Cederqvist et al. Version management with cvs. Free Software Foundation, 1993.

[3] Message Passing Interface Forum. MPI: A message-passing interface standard. *International Journal of Supercomputer Applications*, 8, 1994.

[4] Richard Stallman. Gnu coding standards. Free Software Foundation, 1998.

[5] The GNU Xemacs text editor. `www.gnu.org`. Free Software Foundation, 1998.

A Source Examples

A.1 Makefile Example

```
#
#
#.....................................................................
#...Makefile                                                        ...
#...                                                                ...
#...Purpose                                                         ...
#...=======                                                         ...
#...Creates architecture dependent includes from the c include files ...
#...                                                                ...
#...Also, does "cvs status -v ../README" and gets first occurrence of the ...
#...version which it places inside master_ups.h (and hence in upsf.h) ...
#...                                                                ...
#...Targets                                                         ...
#...=======                                                         ...
#...all:         creates include files                              ...
#...clean:       removes temp files                                 ...
#...clean_n_all: first cleans then creates includes                 ...
#...                                                                ...
#.....................................................................

#........................
#...Variable Descriptions...
#........................
DIRECTORY_NAME = include   # name used for echo statements
UPDATED_FILE =   updated   # name of the updated file

#................
#...Include Files...
#................
include ../make.inc

#.....................................................................
#...all:         creates include files                              ...
#.....................................................................
all::
        @ echo "--gmake $@ ($(DIRECTORY_NAME))--"
        ./sync_f_include.pl $(ARCH) $(DEFINES)

#.....................................................................
#...clean:       removes temp files                                 ...
#.....................................................................
clean:
        @ echo "--gmake $@ ($(DIRECTORY_NAME))--"
        - rm -rf $(MACH) $(OBJS)

#.....................................................................
#...clean_n_all: first cleans then creates includes                 ...
```

```
#... 1) clean: removes temp files                                    ...
#... 2) all:   creates include files                                 ...
#.....................................................................
clean_n_all: clean all
        @ echo "--gmake $@ ($(DIRECTORY_NAME))--"
        @ echo "things have been cleaned and built"
```

A.2 C Include File Example

```
#ifndef UPSP_GS_H
#define UPSP_GS_H

#ifdef __cplusplus
extern "C"
{
#endif

/* ========================================================================== */
/* UPS Gather/Scatter private include file.                                   */
/* ========================================================================== */

  /* -------------------------------- */
  /* global variables for gs component */
  /* -------------------------------- */
  typedef struct
  {
    int
      initialized,       /* if gs is initialized already */
      io_pe;             /* io pe (should be in aa component) */

    int                  /* For UPS_GS_Setup. */
    *PE_index_count,     /* how many indices belong to each pe */
      *PE_recv_count,    /* indices to be received */
      *PE_send_count,    /* indices to be sent */
      number_gs_structs; /* the current number of gs structs in use */

    int
    request_status_count; /* count of items that below buffers can hold */

    void
    *recv_request,       /* recv request buffer */
      *recv_status,      /* recv status buffer */
      *send_request,     /* recv request buffer */
      *send_status;      /* send status buffer */

    int                  /* For UPS_GS_Collate and UPS_GS_Distribute. */
    *counts,             /* how many each pe will be send/recv 'ing */
      *displs;           /* offset to pe's data */

    upsp_gs_id_data
    *first_gs_id_data,   /* points to first gs communication struct */
      *last_gs_id_data;  /* points to last gs communication struct */


  } ups_gs_globals;
  UPS_EXTERN ups_gs_globals upsp_gs;
```

```
#include "upsi_gs.h"

/* ---------------------------------------------- */
/* remove typesafe linkage if compiling under c++ */
/* ---------------------------------------------- */
#ifdef __cplusplus
}
#endif

#endif /* UPSP_GS_H */
```

A.3 C Source Example

```
***** Note: required fields will be marked by ‘‘<<<required>>>’’ *****



#include "ups.h"
#include "upsp_aa.h"

<<<required one line per variable>>>
int UPS_CM_Move(
                const void       *x,
                void             *y,
                int              count,
                UPS_DT_Datatype  datatype,
                UPS_AA_Operation reduce_op
                )
{

<<<required begin header comment delimiter>>>
/* ============================================================================

<<<required Name>>
   Name
   ====
   UPS_CM_Move

   Package
   =======
   cm <<1 word or, if not given, derived from second part of NAME above>>

<<<required Purpose>>>
   Purpose
   =======
   Brief description of the routine

<<<required Arguments (intent, type, and description)>>>
   Arguments
   =========
   x          Intent:       in
              C       type: const void*
              Fortran type: user_choice {<<dimension spec:possible count spec>>}
              The starting address of the local values.

   y          Intent:       out
              C       type: void*
              Fortran type: user_choice {<<dimension spec:possible count spec>>}
              The starting address of the result of the reduction.
              Note that this cannot be the same memory location as {\tt x}.
```

```
count      Intent:       in
           C       type: int
           Fortran type: UPS_KIND_INT4 {0}
           The number of local values. That is, the length of {\tt x}
           in terms of the input datatype.

datatype   Intent:       in
           C       type: int
           Fortran type: UPS_KIND_INT4 {0}
           The type of the local data.
           See the User Guide for datatypes relevant to the
           reduction operation. Note the special datatypes
           that must be used for the {\tt MAXLOC} and {\tt MINLOC}
           operations.

reduce_op  Intent:       in
           C       type: UPS_AA_Operation
           Fortran type: integer
           The operation to be performed upon the local data. \\
           Please see UPS_AA_Operation
           (section \ref{constant\_UPS\_AA\_Operation}
           page \pageref{constant\_UPS\_AA\_Operation}) for a listing of
           the possible operations.

mask       Intent:       in
           C       type: (na) see UPS_CM_Movem
           Fortran77 type: (na) see UPS_CM_Movem
           Fortran type: (optional:UPS_CM_Movem) UPS_KIND_INT4 {x}
           Mask for input variable x.

ierr       Intent:       out
           C       type: (na) int (function return value)
           Fortran type: integer
           Return status. Values other than 0 indicate an error.

UPS_MEM_NUM_GUARD_BLOCKS  Intent: in
                          C type: (na) int Environment Variable
                          Change the default value of the number of
                          additional blocks allocated for guard bytes before
                          and after the requested space.
                          By default, this value is 1 and the size of
                          a block is the size needed to align memory
                          (typically the size of a C double).
                          Remember to propagate environment variables to
                          all processes.  May not be modified during run.

Discussion
==========
(LaTeX form additional in depth discussion can go here.)
```

```
   Examples
   ========
   (LaTeX form examples can go here.)

<<<required Return Values>>>
   Return Values
   =============
   Returns UPS_OK if successful.

   Errors
   ======
   (any discussion of specific error return values here)

   Versions
   ========
   (any discussion about other version [mpi/pvm/serial, cvs version, ...])

   See Also
   ========
   (whitespace separated list of user C interface routine names)

<<<required Program Flow>>>
   Program Flow
   ============
   1) gather data into temporary buffer depending upon process info
   2) move data in reverse process order so process 0 done last.
      This is done because we need so save process 0 for input/output.
   3) (other statements)

<<<required end header comment delimiter>>>
   ========================================================================== */

<<<required Local Declarations (variables w/ comment on separate line)>>>
  /* ------------------ */
  /* Local Declarations */
  /* ------------------ */

  double
    alpha,          /* temporary storage variable        */
    x_out[10],      /* starting data                     */
    *y_out,         /* intermediate data                 */
    zeta,           /* temporary storage variable        */
    z_out[10];      /* final data                        */

  float
    precision;      /* stores the precision of the answer */

  int
    i,              /* counter variable                  */
    ierr = UPS_OK;  /* error return variable             */
```

```
<<<required Executable Statements>>>
  /* -------------------- */
  /* Executable Statements */
  /* -------------------- */

  /* ------------- */
  /* Assertions In */
  /* ------------- */
<<<check for package initialization>>>
  UPSP_ER_ASSERT( upsp_cm.initialized,
                  "UPS not initialized - call UPS_AA_Init()",
                  UPS_ERROR_AA_INIT );

<<<check for argument validity>>>
  UPSP_ER_ASSERT( x && y,
                  "Invalid addresses",
                  UPS_ERROR_AA_BAD_PARAMETER );
  UPSP_ER_ASSERT( count >= 0,
                  "Invalid parameter",
                  UPS_ERROR_AA_BAD_PARAMETER );

  /* ----------------------- */
  /* variable initialization */
  /* ----------------------- */
  x[0] = 1;
  y[0] = 2;
  z[0] = 3;

<<<when allocating memory, use upsp_ut.h macros>>>
  UPSP_UT_MEM_MALLOC( 10*sizeof( double ), &(y) );

<<<required long code block comment saying what is to be done>>>
  /* ------------------------------------------------------ */
  /* loop to set new values based upon input parameters.    */
  /* this is done to allow easier access to other routines  */
  /* ------------------------------------------------------ */
  for ( i = 0; i < 10; i++ )
<<<required ``{'' on lines by themselves>>>
    {
      /* some code */
      ierr = upsp_aa_foo( x, y, z );
<<<required error check code>>>
      UPSP_ER_ASSERT( !ierr,
                      "upsp_aa_foo",
                      UPS_ERROR_CM_MOVE );
    }

<<<required long code block comment saying what was done>>>
  /* ------------------------------------------------------- */
  /* DONE: loop to set new values based upon input parameters. */
  /* ------------------------------------------------------- */
```

```
  /* ------------------------------------------------- */
  /* read in data file for movement technique to be used */
  /* ------------------------------------------------- */

  /* finish routine */

}
```